Int J Digit Libr (2014) 15:27–51
DOI 10.1007/s00799-014-0125-z

A case study on propagating and updating provenance information using the CIDOC CRM

Christos Strubulis · Giorgos Flouris · Yannis Tzitzikas · Martin Doerr

Received: 8 January 2013 / Revised: 6 August 2014 / Accepted: 7 August 2014 / Published online: 29 August 2014

Abstract Provenance information of digital objects maintained by digital libraries and archives is crucial for *authenticity assessment*, *reproducibility* and *accountability*. Such information is commonly stored on metadata placed in various Metadata Repositories (MRs) or Knowledge Bases (KBs). Nevertheless, in various settings it is prohibitive to store the provenance of each digital object due to the high storage space requirements that are needed for having complete provenance. In this paper, we introduce provenance-based inference rules as a means to complete the provenance information, to reduce the amount of provenance information that has to be stored, and to ease quality control (e.g., corrections). Roughly, we show how provenance information can be *propagated* by identifying a number of basic inference rules over a core conceptual model for representing provenance. The propagation of provenance concerns fundamental modelling concepts such as *actors, activities, events, devices* and *information objects*, and their associations. However, since a MR/KB is not static but changes over time due to several factors, the question that arises is how we can satisfy update requests while still supporting the aforementioned inference rules. Towards this end, we elaborate on the specification of the required add/delete operations, consider two different semantics for deletion of information, and provide the corresponding update algorithms. Finally, we report extensive comparative results for different repository policies regarding the derivation of new knowledge, in datasets containing up to one million RDF triples. The results allow us to understand the tradeoffs related to the use of inference rules on storage space and performance of queries and updates.

Keywords Digital data · Knowledge base · Inference rules · Derivation rules · Provenance · Knowledge evolution · Storage space

C. Strubulis · G. Flouris · Y. Tzitzikas (✉) · M. Doerr
Institute of Computer Science, FORTH-ICS, Heraklion, Crete, Greece
e-mail: tzitzik@ics.forth.gr

C. Strubulis
e-mail: strubul@ics.forth.gr

G. Flouris
e-mail: fgeo@ics.forth.gr

M. Doerr
e-mail: martin@ics.forth.gr

Y. Tzitzikas · C. Strubulis
Computer Science Department, University of Crete, Heraklion, Crete, Greece

1 Introduction

The amount of digital objects (publications, datasets, artworks) that libraries and archives have to maintain constantly increases [3]. Digital libraries and archives should be able to assess the *authenticity* [51], the *accountability* [5] and the *reproducibility* [53] of the archived digital objects. This is of paramount importance for e-Science. It is therefore important to document and preserve the history of digital objects, i.e., their *provenance*, and this is usually done by extra metadata. To store such metadata, more and more Digital Libraries (DLs) rely on Semantic Technologies. This is true for particular DLs (e.g., Europeana[1]) or for tools and systems which are used for setting up DLs (e.g., Fedora[2]).

In a naive view, the provenance of a digital object can be seen as a record specific to it of the events and their contexts that have contributed or had significant influence on its content. However, digital objects do not undergo "changes" as

[1] http://www.europeana.eu/.

[2] http://www.fedora-commons.org/.

material items, which sum up to a cumulative effect, but each modification leaves behind the original version, which may or may not be reused in another context. Hence any realistic creation process of digital content gives rise to a set of digital items, temporary or permanent, connected by metadata forming complex graphs via the individual processes contributing to it. The provenance of a single item is the graph of all "upstream" events until the ultimate empirical capture (measurement), software simulation run or human creative process.

In a production environment, often controlled by a workflow system, there are no clear a priori rules regarding which data item will be permanent. Interactive processes of inspection of intermediate results and manual interventions or changes of processing steps may corrupt any preconceived order of events. Therefore, one should explicitly store the provenance of each digital object. However, in cases like empirical 3D model generation, where tens of thousands of intermediate files and processes of hundreds of individual manual actions are no rarity, it is prohibitive to register for each item its complete history because of the immense repetition of facts between the files: on one side, the storage space needed would be blown up by several orders of magnitude, and on the other, any correction of erroneous input would require tracing the huge proliferation graph of this input.

To tackle this problem, in this work we show how *inference rules* can be used for *propagating* provenance information. These rules can complete the initial ingested provenance information such as that if a camera participates to an event (e.g., a shooting event), then its lens also participates to the same event. This completion is *dynamic*, i.e., the propagated information is not explicitly stored. Consequently this reduces the space requirements, as well as the search space for detecting and correcting possible errors. In addition, it saves space, which may be important in applications where storage space is more expensive than processing power (e.g., when the data must be sent over a slow network). A further advantage of this method is that people can more easily understand and perform changes on the ingested data, than on the one produced by the application of inference rules.

Given that our approach is more valuable over dynamic data, we place emphasis on addressing the problem of evolution, in the presence of inference rules for dynamic provenance propagation. To the best of our knowledge, this is not covered by other provenance-related papers. For instance, [19,35] consider minimization techniques on already stored information, which is in contrast to our approach that proposes a dynamic derivation of new knowledge which does not have to be stored. Furthermore, our work is complementary to the works that infer provenance dependencies between data such as [15,23,44,45].

For representing provenance, we adopt a core conceptual model that contains fundamental (for provenance) modelling concepts such as *actors, activities, events, devices, information objects* and their associations. Over this model we identify three basic custom inference rules which occur frequently in practice. Of course, one could extend this set according to the details and conventions of the application at hand.

However, we should consider that a knowledge base (either stored in a system or composed by various metadata files) changes over time. The question that arises is how we can satisfy update requests while still supporting the aforementioned inference rules, for instance: how one can delete provenance information that has been propagated, i.e., information that is produced by inference rules and is not explicitly stored in the repository?

To tackle the update requirements we propose three operations, namely *Add*, *Disassociate*, and *Contract*, and discuss their semantics along with the required algorithms. The last two operations concern the deletion of information and are founded on the related philosophical viewpoints, i.e., *foundational* and *coherence* semantics. These viewpoints actually differ in the way they value the inferred knowledge in comparison to the explicitly ingested one. This is novel in the literature, because existing works (e.g., the approaches [34,37,43]) consider only one of the viewpoints and only the standard RDF/S inference rules (not custom inference rules).

Having specified provenance propagation, and provenance update, the next step is to evaluate this approach. For this reason we report comparative results for alternative repository policies regarding the storage of new (derived) knowledge. The policy proposed here does not store any inferred knowledge. This is in contrast to existing approaches (discussed in Sect. 4), where both the explicitly ingested and the inferred information is stored.

The objective of this evaluation is to enable us to understand the trade off among space usage, query performance and update performance for these policies. To this end, we conducted experiments that measure the storage space before and after the application of the inference rules, and the impact of inferencing on the performance of queries and updates. For these experiments we used real-world data and synthetic data that exhibit features observed in typical real-world datasets. In brief, the results showed that our suggested policy (i.e., storing only the explicit provenance information) significantly reduces the storage space requirements of provenance information while introducing only a small performance overhead for queries and updates.

The rest of this paper is organized as follows: Sect. 2 discusses the provenance model that we consider in this work and continues with a motivating example; Sect. 3 introduces the provenance inference rules; subsequently, Sect. 4 analyses how the explicit and inferred knowledge can be stored and the assumptions considered for the rest of the paper; Sect. 5 details on the knowledge evolution requirements and their

interplay with the inference rules; Sect. 6 explains the algorithms defined for the knowledge evolution; Sect. 7 presents the experimental evaluation of this work; Sect. 8 elaborates on the applicability of our work; Sect. 9 compares related work to our approach, and finally Sect. 10 concludes the paper and identifies issues that are worth further research. Extra material including the signatures of the proposed update operations and their algorithms is given in the Appendix.

This paper extends the ideas first presented at [57]. In comparison to that work, this paper details the implementation, reports extensive and comparative experimental results over real and synthetic datasets, discusses the applicability of our approach (working assumptions and repository policies) and contains a detailed discussion of related work.

2 Provenance model and motivating example

2.1 Provenance model

There are several models for representing provenance, such as OPM [44], ProvDM [45] and Dublin Core [2]. All of them contain some basic concepts like *actors, activities, events, devices, information objects* and associations between them (e.g., actors carrying out activities or devices used in events). In this work we consider a basic model shown in Fig. 1. It consists of six concepts and nine associations. It is actually part of CIDOC CRM (ISO 21127:2006) [25], which is a core ontology describing the underlying semantics of data schemata and structures from all museum disciplines and archives.

CIDOC CRM can be applied for scientific data because scientific data and metadata can be considered as historical records: scientific observation and machine-supported processing are initiated on behalf of and controlled by human activity. Things, data, people, times and places are related by events, while other relations are either deductions from events or found by observation. Furthermore, CIDOC CRM has been extended, resulting in CRMdig [59][3], for better capturing the modelling and query requirements of digital objects. In numbers, CIDOC CRM contains 86 classes and 137 properties, while its extension CRMdig currently contains an additional set of 31 classes and 70 properties.

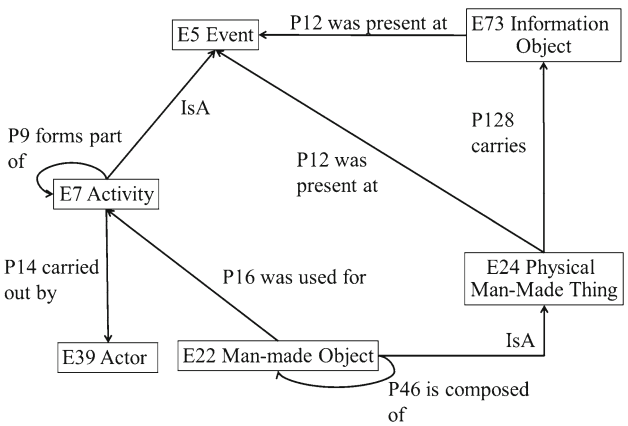

Fig. 1 Part of CRM Dig

As already mentioned, it is enough to consider only the small part shown at Fig. 1, since only that part is involved in the inference rules which are introduced in Sect. 3. Even though we focus on the CIDOC CRM model, the approach is more general and can be applied to any RDF-based annotation model. Furthermore, this set of rules is only indicative and one can follow the logic of our approach to define extra rules for query or update operations.

The shown classes and properties are described in detail in CIDOC CRM's official definition [1]. In brief, the properties *P46 is composed of* and *P9 forms part of* represent the part-hood relationships of man-made objects (i.e., instances of the *E22 Man-made Object* class) and activities (i.e., instances of the *E24 Physical Man-Made Thing* class), respectively. According to their semantics, these properties are transitive, reflexive and antisymmetric.

The property *P14 carried out by* describes the active participation of actors (i.e., instances of the *E39 Actor* class) in activities and also implies causal or legal responsibility. In this respect, actors could have participated in some part of an activity, and not necessarily in all of it.

The property *P16 was used for* describes the use of objects in a way essential to the performance of an activity. It also indicates that the related objects were used as whole objects, and all the parts of them were involved in the execution of an activity.

Finally, immaterial items (i.e., instances of the *E73 Information Object* class) are related to physical carriers (i.e., instances of the *E24 Physical Man-Made Thing* class) via the *P128 carries* property and can be present to events via the *P12 was present at* property.

2.2 Motivating example

The Mars Science Laboratory of NASA launched in 2011 a new rover called *Curiosity* to explore the red planet. *Curiosity* contains a device called *MastCam* (see Fig. 2) designed to take and store thousands of color images and several hours of color video footage of the Martial terrain [47]. However, this digital datum is meaningless without some provenance information, stating, for example, where and when an image (or video) was taken, the parameters and calibration of the *MastCam* during the shot, etc. Completing this knowledge

[3] It was initially defined during the EU Project CASPAR (http://www.casparpreserves.eu/) (FP6-2005-IST-033572) and its evolution continued during the EU Project IST IP 3D-COFORM (http://www.3d-coform.eu/).

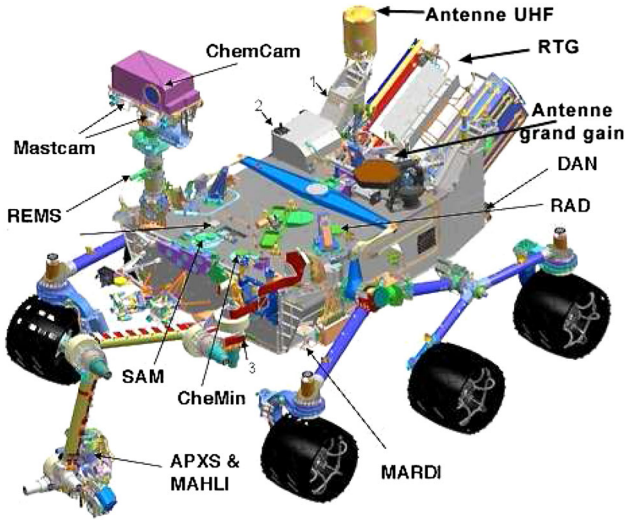

Fig. 2 Curiosity Mars Rover (extracted from NASA's site [47])

would require both the information of usage and its parameters to be associated with each part, having multiple references of the same information, but in relation with different parts, and thus replicating that information. For example, if the *MastCam* was used for a certain shot of the Martian terrain, then its parts were also used for said shot. Our target is to avoid such replication by inferring such knowledge using reasoning rules.

3 Provenance inference rules

3.1 The inference rules

The proposed inference rules concern the classes referred previously, also shown in Fig. 1, and three binary relations, namely *carriedOutBy (Activity, Actor)*, *wasUsedFor (Device, Activity)*, and *wasPresentAt (InformationObject, Event)* which correspond to the associations *P14 carried out by*, *P16 was used for* and *P12 was present at*, respectively. Note that other associations are also related to these rules such as the *P9 forms part of* (whose transitivity could also be expressed by a rule). The rules are defined as:

- R1: If an actor has carried out one activity, then he has carried out all of its subactivities.
- R2: If an object was used for an activity, then all parts of the object were used for that activity too.
- R3: If a physical object that carries an information object was present at an event, then that information object was present at that event too.

More formally, the above three rules can be encoded into first-order logic (FOL), as a general basis for further implementations to other languages such as SWRL or Datalog:

- R1:

$$\forall x, y, z(\text{formsPartOf}(y, x) \wedge \text{carriedOutBy}(x, z) \rightarrow \text{carriedOutBy}(y, z)) \quad (1)$$

- R2:

$$\forall x, y, z(\text{isComposedOf}(x, y) \wedge \text{wasUsedFor}(x, z) \rightarrow \text{wasUsedFor}(y, z)) \quad (2)$$

- R3:

$$\forall x, y, z(\text{carries}(x, y) \wedge \text{wasPresentAt}(x, z) \rightarrow \text{wasPresentAt}(y, z)) \quad (3)$$

The considered inference rules are based on the semantics of the involved concepts/properties, as described in the scope notes of CIDOC CRM [1] and discussed in Sect. 2. In the sequel we provide examples for each rule. In the forthcoming figures we do not show the transitivity-induced properties *P46 is composed of* and *P9 forms part of*. The information that is inferred by each rule is depicted by *dotted lines* in all figures. It should be noted that in this work we consider only positive facts stored in the respective repository and not rules with negation. In this regard, we assume default reasoning and that negative facts result to the deletion of information as an update request (Sect. 5.2).

3.2 Rule 1: carriedOutBy (Activity, Actor)

The *MastCam* consists of two cameras with different focal lengths and different science color filters: (a) the *MastCam-100* having 100 mm focal length and (b) the *MastCam-34* with 34 mm focal length. The manufacturing process of those devices can be preserved by modelling a hierarchy of assembly activities. As a result, we can have the superactivity *Cameras Assembly* and the assembly subactivities of *MastCam, MastCam-100* and *MastCam-34*.

Another crucial information to be preserved is the person, organization or laboratory which was responsible for carrying out these activities. In this example, we record that the *NASA Laboratory* was the actor responsible and associate *Cameras Assembly* with the former using *P14 carried out by*.

Even though the above subactivities have different recorded metadata, the information that the initial actor was the *NASA Laboratory* is desired to be preserved along the hierarchy of activities. This is accomplished by rule R1. Figure 3 shows the edges (represented by dotted lines) which are inferred by the rule.

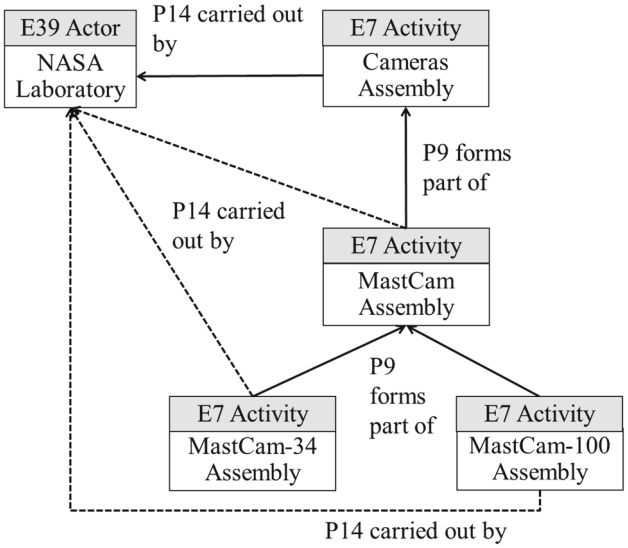

Fig. 3 Example of rule R1

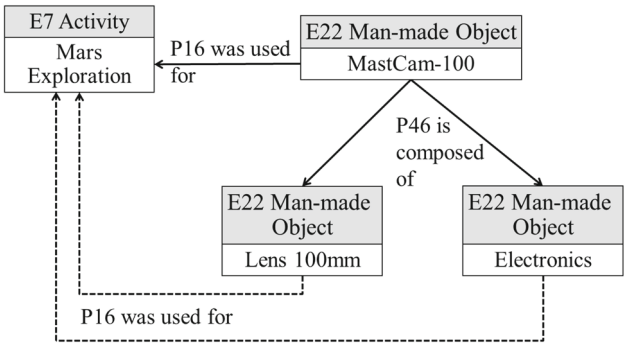

Fig. 4 Example of rule R2

3.3 Rule 2: wasUsedFor (Device, Activity)

We can extend the decomposition of *MastCam-100* by modelling its parts such as its lens and other electronic. Moreover, since the camera and its parts were used for the exploration of Mars, this information could also be stored. Figure 4 illustrates an indicative modelling of such a composition along with the inferences. With rule R2 we can infer that both devices (i.e., *Lens 100mm* and *Electronics*) as parts of *MastCam-100* were used in the activity of Mars exploration.

3.4 Rule 3: wasPresentAt (InformationObject, Event)

Each *MastCam* of *Curiosity* can acquire images of high resolution including 720-p high-definition video. These data could be modelled as an information object being carried by an internal buffer which was present at the event of Mars exploration (see Fig. 5). According to rule R3 both the buffer and the images were present at the referred event. The inference is reasonable since without the presence of the buffer any digital data would not exist.

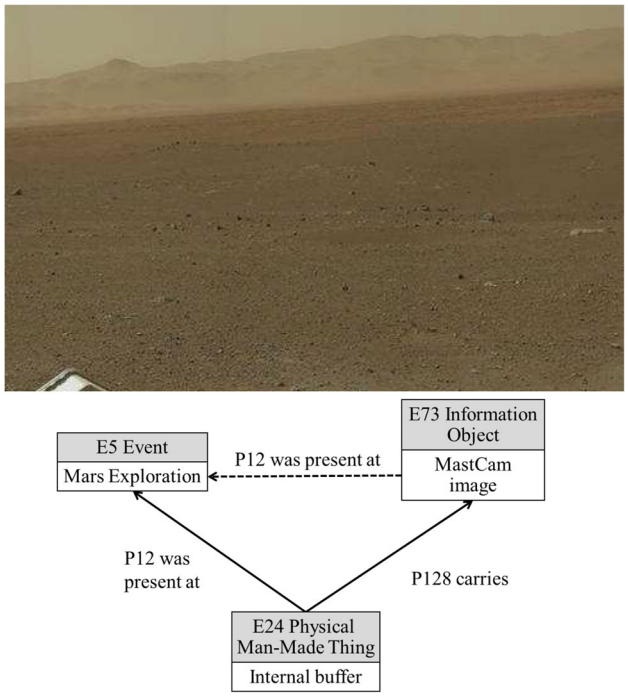

Fig. 5 A high-resolution color mosaic *MastCam*, (*up*, NASA's press release) example of rule R3 (*down*)

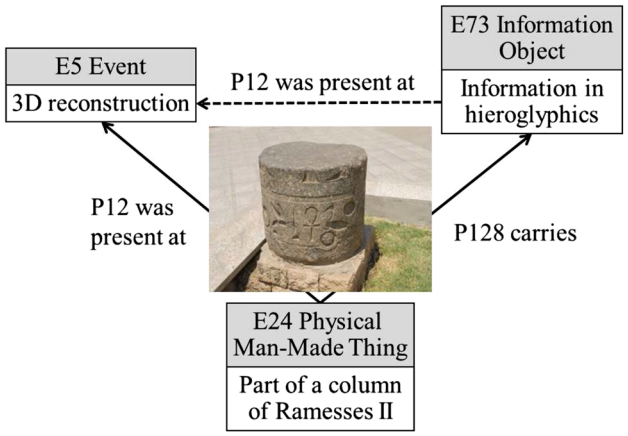

Fig. 6 Example of rule R3 from the cultural heritage domain

Other examples can also be found to realize the extent and applicability of our work to a variety of domains related to digital preservation. One of such examples might be the 3D reconstruction from images used in archaeology to digitize and model archaeological exhibits. For instance the exhibit shown in Fig. 6, which is part of a column of Ramesses II, could be modelled as a physical man-made thing being present at the event of 3D reconstruction. Moreover, the information represented by the carved hieroglyphics could be modelled as an information object (see Fig. 6).

According to rule R3 if that part was present in an event, then that information was also present at that event. Rule R3 infers the presence of the latter because the part of the column of Ramesses II was also present at that event. The inference is reasonable because the information was carved in hieroglyphics when the column was built, thus the information in hieroglyphics coexists with the part of a column which carries it, and this coexistence implies their common presence at events.

3.5 Synopsis

In this section we presented a set of inference rules, to satisfy our primary motivation of this study regarding the propagation of provenance information and the derivation of new associations (i.e., new knowledge) based on logical assumptions. We focused on these three rules as they frequently occur in practice and the related classes and associations are fundamental for modelling provenance. Of course, one could extend this set according to the details and conventions of the application at hand.

4 Inference rules and repository policies

Digital preservation approaches heavily rely on the existence and curation of metadata, and currently well-defined frameworks, such as the Resource Description Framework (RDF) [38], are increasingly used for expressing them. To interpret these metadata within or across user communities, RDF allows the definition of appropriate schema vocabularies (RDF/S) [16].

In RDF, the metadata are formed in triples (of the form *subject*, *property*, *object*), asserting the fact that *subject* is associated with *object* through *property*, and most often these facts are connected forming complicated graphs [13].

RDF/S is used to add semantics to RDF triples, by imposing inference rules (mainly related to the transitivity of subsumption relationships), which can be used to entail new implicit triples (i.e., facts) that are not explicitly asserted. Note that standard generic inference rules are different from the custom inference rules (R1, R2, R3) defined above, and they have different usage, even though it is possible that standard and custom inference rules may have similar structure.

In this section, we study various repository storage policies regarding whether the inferred provenance information is stored or not and can be used in conjunction with the inference rules (either the custom or the RDF/S ones).

We shall use the term KB to refer to a Knowledge Base in the logical sense, composed of the contents of several metadata files which can either be stored in RDF databases (i.e., triple stores/repositories) [48] or in the rapidly growing Linked Open Data (LOD) cloud [12]. We shall use K to refer to the set of RDF/S triples of a KB. Furthermore, hereafter we assume that K denotes the set of triples as produced by the adopted metadata schemas and ontologies and the *ingested* facts ("raw data" yielded by manual or automated processes) without any post-processing.

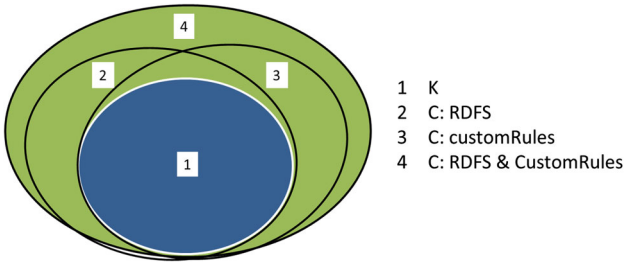

Fig. 7 Repository policies

We shall use $C(K)$ to refer to the *closure* of K, i.e., the data that are produced from the data in K using the considered inference rules. Note that $C(K)$ can be defined with respect to the standard inference rules for RDF/S and/or other custom rules (e.g., like those that we introduce in this paper) and it also includes K. From a visual perspective, in the examples of our rules (Sect. 3), $C(K)$ includes both the dotted and plain associations, while K includes only the latter.

A repository policy could be to keep stored either K or $C(K)$ (see Fig. 7 for an overview of policies in venn diagram representation). One could consider the option of storing K itself. Note that K usually contains little or no redundancy, leading to a near-optimal space usage while avoiding the overhead of searching for, and eliminating, redundancy. Moreover, in some settings, the explicit information in K has a different value from the implicit one in $C(K)$, and the distinction between the two must be kept clear.

The option of storing $C(K)$ is optimal with respect to query efficiency, because all the information is stored and can be efficiently retrieved, but has increased space requirements and some overhead in update and maintenance operations.

The rest of this paper assumes (unless explicitly mentioned otherwise) that K is stored. In our experimental evaluation (Sect. 7) we will also consider the option of storing $C(K)$ and compare the ramifications of this option in terms of storage, query time and update time, as opposed to the option of storing K.

5 Provenance inference rules and knowledge evolution

A KB changes over time, i.e., we may have requests for adding or deleting facts due to external factors, such as new observations [28]. Satisfying update requests while still supporting the aforementioned inference rules is a challenging issue, because several problems arise when updating knowledge taking into account rules and implicit facts. These

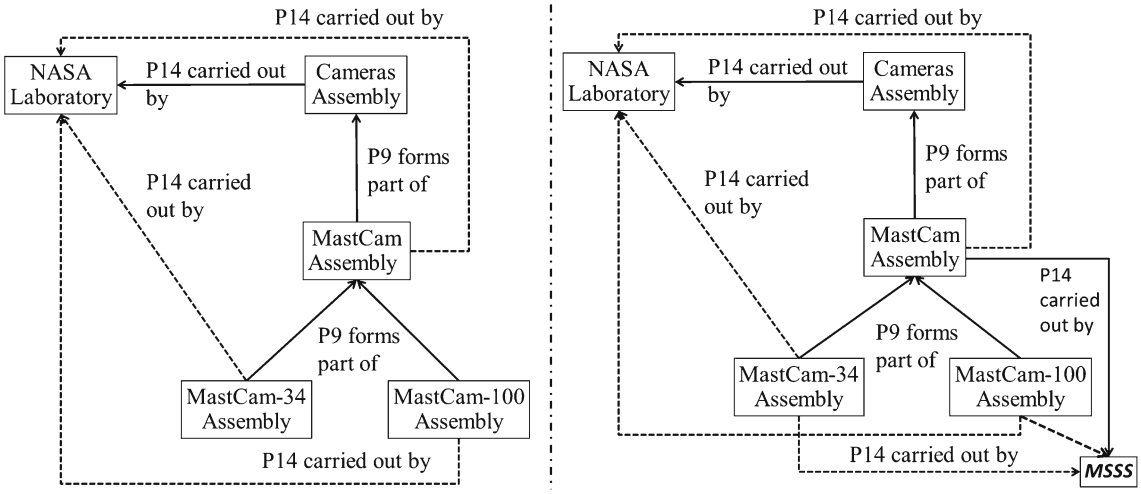

Fig. 8 Initial state of the KB (*left*) and state of the KB after the addition (*right*)

problems will be described in more detail below using a running example. For each update operation, we describe the KB's states (through figures) and explain the challenges related to the update process due to the existence of the inference rules.

So, let us consider a KB that contains the rule R1's example with the activities of *Cameras Assembly* that were carried out by the *NASA Laboratory*. The initial state of the KB is demonstrated in Fig. 8 (left), where inferred associations are illustrated (as usual) by dotted lines. We will consider three update operations: *addition, disassociation, contraction*.

5.1 Addition of information

The *addition* operation performs the insertion of a new triple into the KB. Since there are no validity rules or negation, the addition of a triple cannot lead to a contradiction (as, e.g., in the case of [28] or in the generic belief revision literature [31]). Therefore, addition is a simple operation consisting only of the straightforward addition of the required information (explicit triples) in our KB. Note that if we had taken into account a repository policy that stores its closure ($C(K)$), we would have to apply additional operations after the addition to compute and store the new $C(K)$.

Figure 8 visualizes the process of adding a carried out by relationship. In particular, Fig. 8 shows a request for adding a new actor to the subactivity of the *MastCam Assembly*, namely that the *Malin Space Science Systems*, for short *MSSS*, which is a company designing and building such cameras, is also the actor of *MastCam Assembly*. In the right part of Fig. 8 we show the result of this addition; the node *MSSS* is shown at the bottom right corner of the figure. Observe that *MSSS* has not only been associated with the activity *MastCam Assembly*, but also, due to rule R1, has been implicitly associated with the subactivities *MastCam-34 Assembly* and *MastCam-100 Assembly*. Note that we do not have to explicitly add those implicit triples.

5.2 Deletion of information

The addition of information is used in practice more often than deletion. Deletion is usually a consequential operation taken to amend some kind of conflict. However, this is not always the case. For example, deletion should be employed whenever we lose confidence in a previous observation or measurement, or when some erroneous fact was placed in our KB (e.g., when we realize that a certain object was not used for a particular activity).

The problem when dealing with the deletion of information is more complex than the addition. The reason is that, due to the inference rules, we cannot succeed by simply deleting the required information, as the deleted information may re-emerge as a consequence of the application of the inference rules. Thus, it is often the case that additional information should be deleted, along with the one explicitly requested. This raises the additional challenge of avoiding losing (deleting) any more knowledge than necessary. We will visualize how we address this problem using our running example below.

Consider an update request stating that the association of *NASA Laboratory* with the activity *MastCam-34 Assembly* through property *P14 carried out by* must be deleted. The rising question is whether we should delete the association of *NASA Laboratory* with *MastCam-34 Assembly* only, or whether we should do the same for other activities also; in other words, we should determine how refuting the fact that the *NASA Laboratory* is responsible for *MastCam-34 Assembly* affects the information that it is responsible for the activities *Cameras Assembly, MastCam Assembly* or *MastCam-100 Assembly* (see Fig. 9).

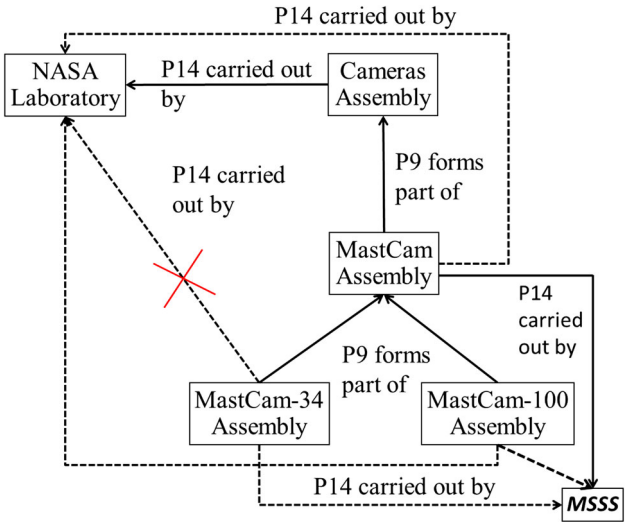

Fig. 9 Deletion of information

Initially, we note that the *NASA Laboratory* should also be disassociated from the responsibility of *MastCam Assembly* and *Cameras Assembly*; failing to do so would cause the subsequent re-emergence of the refuted knowledge (i.e., that the *NASA Laboratory* is not responsible for *MastCam-34 Assembly*) due to inference.

A more complicated issue is whether the *NASA Laboratory* should remain responsible for *MastCam-100 Assembly*: note that this information was originally included just because of the fact that the *NASA Laboratory* was considered responsible for Cameras Assembly, ergo (due to the inference rules), also responsible for *MastCam-100 Assembly*. Once the former information is dropped, as discussed above, it is questionable whether the latter (inferred) information should still remain in the KB, since its "reason for existence" is no longer there. On the other hand, the fact that the *NASA Laboratory* is not responsible for *MastCam-34 Assembly* does not in any way exclude the possibility that it is still responsible for *MastCam-100 Assembly*; therefore, deleting this information seems like an unnecessary loss of knowledge.

To address this issue, one should go deeper and study the related philosophical issues with regard to the epistemological status of the inferred knowledge, and whether such knowledge has the same or different value compared to primary, explicitly provided knowledge (i.e., ingested knowledge). There are two viewpoints in this respect: *foundational theories* and *coherence theories* [32].

Under the *foundational* viewpoint, each piece of our knowledge serves as a justification for other beliefs; our knowledge is like a pyramid, in which "every belief rests on stable and secure foundations whose identity and security does not derive from the upper stories or sections" [55]. This viewpoint implies that the ingested facts (the "base of the pyramid") are more important than other knowledge and that implicit knowledge has no value of its own, but depends on the existence and support of the explicit knowledge that caused its inference.

On the other hand, under the *coherence* theory, no justification is required for our knowledge; each piece of knowledge is justified by how well it fits with the rest of the knowledge, in forming a coherent set of facts that contains no contradictions. In this sense, knowledge is like a raft, where "every plank helps directly or indirectly to keep all the others in place" [55]; this means that all knowledge (implicit or explicit) have the same "value" and that every piece of knowledge (including implicit ones) is self-justified and needs no support from explicit knowledge.

This distinction is vital for effective management of data deletions. When a piece of knowledge is deleted, all implicit data that are no longer supported must be deleted as well under the foundational viewpoint. In our example, this should cause the deletion of the fact that the *NASA Laboratory* is responsible for *MastCam-100 Assembly*. On the other hand, the coherence viewpoint will only delete the implicit data if it contradicts with existing knowledge, because the notion of support is not relevant for the coherence model. Therefore, in our case, the fact that the *NASA Laboratory* is responsible for *MastCam-100 Assembly* should persist, because it does not in any way contradict the rest of our knowledge or cause the re-emergence of the newly deleted information (i.e., that the *NASA Laboratory* is responsible for *MastCam-34 Assembly*).

Instead of positioning ourselves in favour of one or the other approach, we decided to support both. This is done by defining two different "deletion" operations, namely, *disassociation* (foundational) and *contraction* (coherence), that allow us to support both viewpoints. We describe these operations in more detail in the sequel.

5.2.1 Disassociation

As mentioned above, disassociation handles deletion using the foundational viewpoint. In particular, the non-responsibility of the *NASA Laboratory* about *MastCam-34 Assembly* implies that it is no longer responsible for other related activities (i.e., *MastCam-100 Assembly*), since this knowledge is no longer supported by any explicit data. Based on the foundational viewpoint, all such associations must also be deleted, i.e., we should delete the following:

- *(MastCam-34 Assembly, carried out by, NASA Laboratory)*, as requested
- *(MastCam Assembly, carried out by, NASA Laboratory)*, to avoid re-emergence of the deleted knowledge
- *(Cameras Assembly, carried out by, NASA Laboratory)*, to avoid re-emergence of the deleted knowledge, and

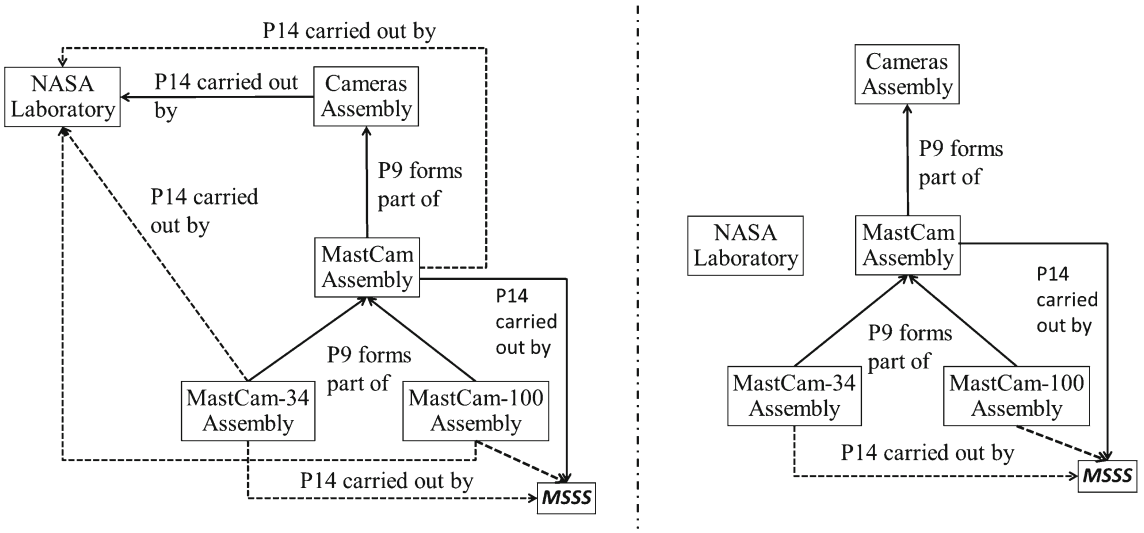

Fig. 10 Initial state of the KB (*left*) and state of the KB after disassociation (*right*)

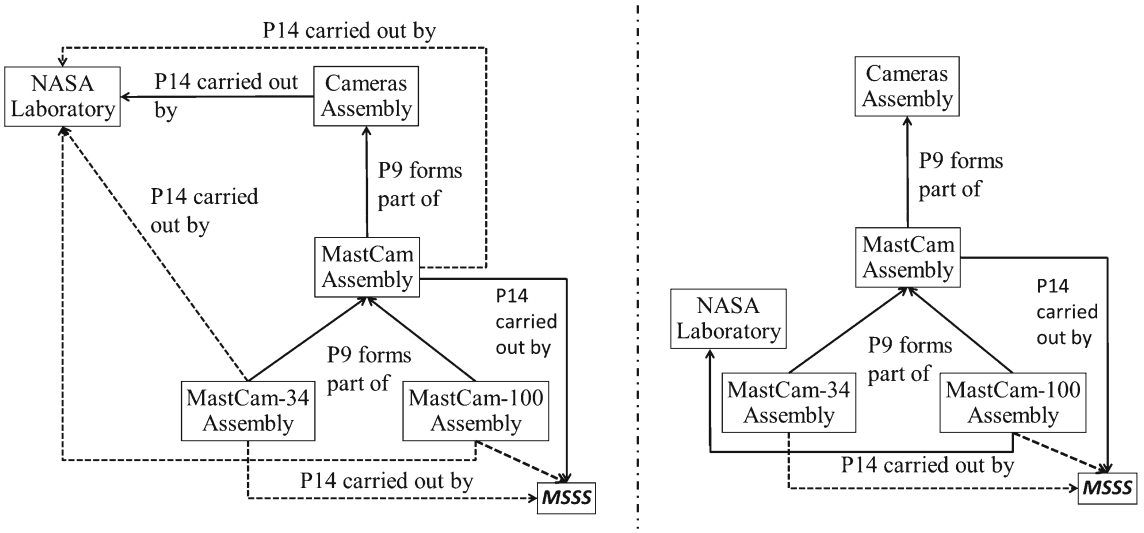

Fig. 11 Initial state of the KB (*left*) and state of the KB after contraction (*right*)

- *(MastCam-100 Assembly, carried out by, NASA Laboratory)*, due to the loss of explicit support.

The right side of Fig. 10 illustrates what information will be stored in the KB after disassociation. Note that, in practice, implicit facts are not stored so need not be deleted; thus, in our case, we only need to delete (*Cameras Assembly, carried out by, NASA Laboratory)*. We will be referring to the above case as *actor disassociation*.

5.2.2 Contraction

Contraction handles deletion using the coherence viewpoint. In particular, this operation assumes that the non-responsibility of the *NASA Laboratory* is only for *MastCam-34 Assembly*. Other activities which are still associated with *NASA Laboratory*, such as *MastCam-100 Assembly*, should persist despite the lack of explicit knowledge to support them. In this case we have to delete only the following:

- *(MastCam-34 Assembly, carried out by, NASA Laboratory)*, as requested,
- *(MastCam Assembly, carried out by, NASA Laboratory)*, to avoid re-emergence of the deleted knowledge, and
- *(Cameras Assembly, carried out by, NASA Laboratory)*, to avoid re-emergence of the deleted knowledge.

The right side of Fig. 11 illustrates what information will be stored in the KB after contraction. Again, implicit facts need not be deleted, so the only actual deletion required is the

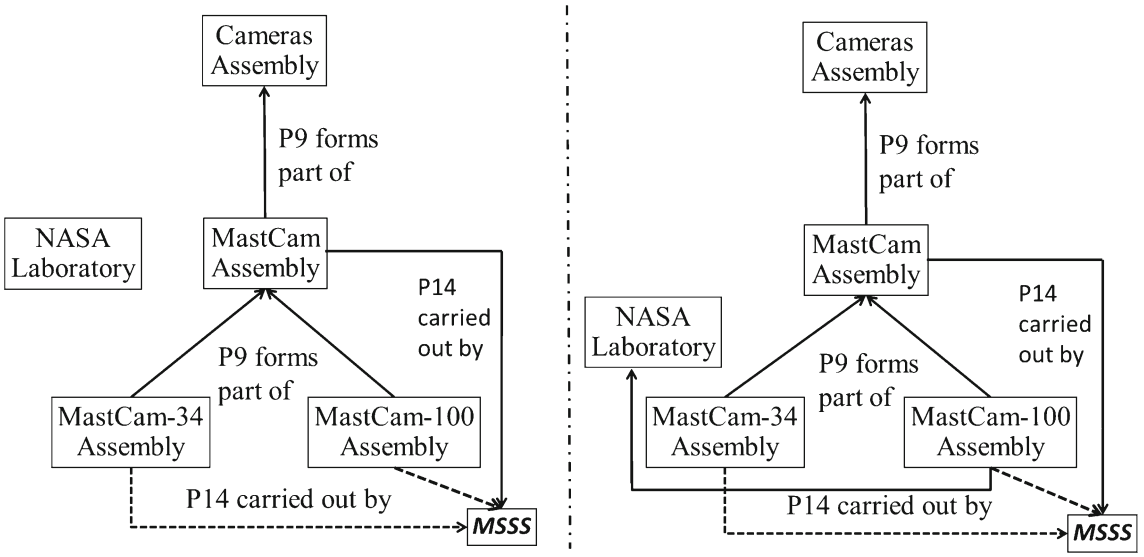

Fig. 12 Actor disassociation (*left*) and actor contraction (*right*)

deletion of *(Cameras Assembly, carried out by, NASA Laboratory)*. For contraction, one should also be careful with the implicit knowledge that is supposed to persist. For example, the fact that the *NASA Laboratory* is responsible for *MastCam-100 Assembly* is not explicitly stored and will be lost along with the deletion of *(Cameras Assembly, carried out by, NASA Laboratory)*, unless we explicitly add it back. Therefore, contraction essentially consists, in this example, by the deletion of *(Cameras Assembly, carried out by, NASA Laboratory)* and the addition of *(MastCam-100 Assembly, carried out by, NASA Laboratory)*. Notice that in contrast to disassociation, the contraction operation preserves the association *(MastCam-100 Assembly, carried out by, NASA Laboratory)* (see Fig. 12). We will be referring to the above case as *actor contraction*.

6 Knowledge evolution: algorithmic perspective

An analysis like that of the previous example can be applied for deriving the exact plan for each update operation. Indicatively, we provide below update plans (algorithms) for three operations:

- AssociateActorToActivity (p:Actor, a:Activity)
- DisassociateActorFromActivity(p:Actor,a:Activity)
- ContractActorFromActivity (p:Actor, a:Activity)

Algorithm 1 takes as input an actor (p) and an activity (a), checks if the information that p is responsible for a already exists in the KB (line 1), and if not, adds that fact as an explicit one in the KB (line 2). This is an easy operation, requiring just the addition of said triple in K.

Algorithm 1 AssociateActorToActivity (p:Actor, a:Activity)

1: **if** an explicit P14 link does not exist between a and p **then**
2: Add an explicit P14 link between a and p
3: **end if**

Algorithm 2 takes the same input (actor p and activity a), but its purpose is to disassociate p from the responsibility for a. Firstly, the requested explicit association has to be removed from the KB (lines 1–3). Secondly, according to the semantics given above, this also requires the deletion of all associations of p with all superactivities of a (lines 4–6). Note that only explicit links need to be removed, because implicit ones do not actually exist in K.

Algorithm 2 DisassociateActorFromActivity (p:Actor, a:Activity)

1: **if** an explicit P14 link exists between a and p **then**
2: Remove the requested P14 link between a and p
3: **end if**
4: **for** each superactivity:superAct of a related to p via the P14 link **do**
5: Remove possible explicit P14 link between superAct and p
6: **end for**

Finally, Algorithm 3 contracts p from the responsibility for a. This requires, apart from the deletion of all associations of p with all superactivities of a (as in disassociation), the preservation of certain implicit associations that would otherwise be lost. At first, any inferred associations between p and any direct subactivities of a must be explicitly added to the KB (lines 1–5). Moreover, additional associations that need to be preserved are stored in Col (lines 9–15); to avoid adding redundant associations, we only consider the

Algorithm 3 ContractActorFromActivity (p:Actor, a:Activity)

```
1: if an explicit P14 carried out by link exists between a or a superactivity of a and p then
2:     for each maximal subactivity:subAct of a do
3:         Execute AssociateActorToActivity (p, subAct)
4:     end for
5: end if
6: if an explicit P14 carried out by link exists between a and p then
7:     Remove the requested P14 carried out by link between a and p
8: end if
9: for each maximal superactivity:supAct of a related to p via the P14 carried out by link do
10:    for each subactivity:subAct of supAct do
11:        if subAct is not superactivity or subactivity of a then
12:            Add subAct to collection: Col
13:        end if
14:    end for
15: end for
16: Execute DisassociateActorFromActivity (p, a)
17: for each maximal activity:act in Col do
18:    Execute AssociateActorToActivity (p, act)
19: end for
```

maximal elements of Col to add the new explicit associations (lines 17–19).

We should clarify that each operation is independent, in the sense that after its execution there is no need for other operations to be performed to complete a particular change. However, their design is modular and thus different operations may use the same algorithms or parts of them. For instance, Algorithm 3 executes Algorithm 2 at line 16 and Algorithm 1 at line 18.

Furthermore, our operations guarantee that the resulting KB will contain (or not contain, depending on the operation) the added/deleted triple, either as an explicit or as an implicit fact, given the existing knowledge and the custom inference rules that we consider. This has been coined as the Principle of Primacy of New Information in the belief revision literature [21]. In addition, our operations preserve as much as possible of the knowledge in the KB under the considered semantics (foundational for the disassociate operation and coherence for the contraction operation). This is known as the Principle of Minimal Change in belief revision terminology.

6.1 Algorithmic complexity

The complexity of the above algorithms is $O(\log N)$ for Algorithm 1, $O(N \log N)$ for Algorithm 2 and $O(N^2)$ for Algorithm 3, where N is the number of triples in K. The above complexities assume that the triples in K are originally sorted (in a preprocessing phase); such a sorting costs $O(N \log N)$. Under this assumption, Algorithm 1 practically needs to check whether a certain fact exists in a sorted table, and if not, to add it, thus the $O(\log N)$ cost.

Algorithm 2 requires the computation of all the superactivities of an activity. To do that, we need to find all the direct superactivities of a (i.e., those connected to a via the *P9 forms part of* property), a process which costs $O(\log N)$; for each such activity, one needs to add it in a sorted collection that contains all the superactivities of a found so far, costing $O(\log M)$, where M is the size of the collection. M can never exceed N, so the total computation time for finding one superactivity of a and adding it in our list is $O(\log N)$. Continuing this process recursively, we will eventually add all superactivities of a, which are at most N, so the process can be repeated at most N times, costing a total of $O(N \log N)$. For each superactivity, we need to determine whether a *P14 carried out by* link to p exists, and if so, delete it; this costs $O(\log N)$ for each superactivity, i.e., $O(N \log N)$ in total (as we can have at most N superactivities). Thus, the combined computational cost for Algorithm 2 is $O(N \log N)$.

Algorithm 3 is more complicated. As in disassociation, we first need to find all superactivities of a, which costs $O(N \log N)$. Then, we need to find the maximal superactivities; this process requires a filtering over the set of all superactivities. In the worst-case scenario, this requires checking each superactivity against all others, costing $O(M^2)$ if the total number of superactivities are M. Thus, in the worst-case scenario (where $M = O(N)$), the cost of finding the maximal superactivities is $O(N^2)$. After that, we need to compute Col (line 7), which is a process identical to the computation of superactivities and requires $O(N \log N)$ time. Finding the maximal elements in Col likewise requires $O(N^2)$, whereas the execution of the disassociation operation is $O(N \log N)$ as explained above. Summing up the above complexities, we conclude that the worst-case computational complexity for Algorithm 3 is $O(N^2)$.

6.2 Additional operations

The *addition* algorithm adds an explicit triple, only if there is no such triple already stored in our KB. In this regard, it should be noted that the addition algorithm is actually following the foundational viewpoint respecting the fact that explicit information is more important than implicit. This can be understood by noticing that if one tries to add a triple that is implied by K, but is not explicit, then the triple will be added (cf. Algorithm 1). This is based on the understanding that it is important to distinguish between explicitly ingested facts and implicit ones, i.e., the discrimination of implicit and explicit triples advocated by the foundational viewpoint. If explicit and implicit triples were considered of the same value (i.e., under the coherence viewpoint), then the addition of a triple that is already implicit would have no effect.

This choice of semantics for the addition operation was made to subsequently allow deletions under the foundational

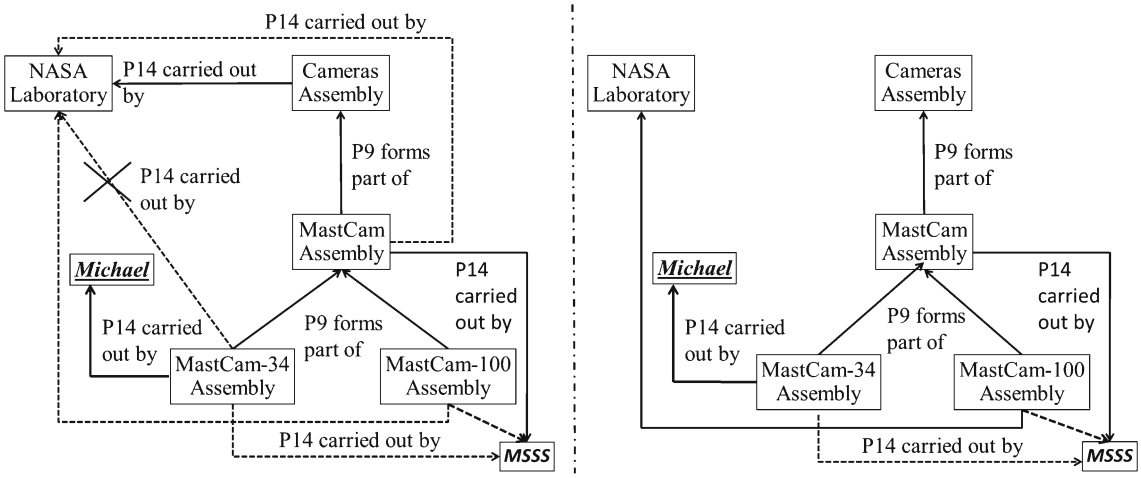

Fig. 13 Actor replacement

viewpoint. Indeed, if one did not add triples that are implicit, then disassociation would produce incorrect results, because (some of) the explicit triples would not be marked as such. At any rate, an addition operator under the coherence viewpoint can be easily implemented, if desired, by just changing the check of line 1 in Algorithm 1 to check for both implicit and explicit triples (rather than just existing ones).

The algorithms for the operations related to the other inference rules, e.g., for *DisassociateActivityFrom MMObject* and *ContractActivityFromMMObject* (which are related to rule R2), can be designed analogously (see Algorithms 11 and 12 in Appendix B). Since hierarchies of devices and parts have the same graph morphology, the respective algorithms follow the same design as the ones of R1 discussed previously. However, some obvious adjustments are essential such as the replacement of the *P14 carried out by* with the *P16 was used for* link and the activities with devices (i.e., man-made objects) which are composed of parts instead of subactivities. Moreover, instead of the actor, we now have the activity that used the specific device.

The operations for rule R3 are not applied on hierarchies since according to the semantics, the respective associations *P12 was present at* and *P128 carries* are not transitive. Thus, they are not so complicated and their algorithms can be easily implemented. For instance, for the case of the foundational viewpoint, the deletion of a *P12 was present at* link between an information object and an event requires the deletion of the respective *P12 was present at* link between the carriers of the former and the latter (see Algorithm 22 in Appendix B). Note that information objects cannot exist and thus be present at events without being carried by physical things. For this reason, we only consider foundational semantics.

For reasons of space, the signatures of the complete set of operations are given in Appendix A of this paper. Using the same mindset, we could also develop algorithms for adding/deleting the transitive relationships used in our model, such as *P9 forms part of*, *P46 is composed of*, etc., as well as for adding/deleting new objects, such as actors, activities, etc.

In addition, one could *compose* the above operations to define more complex, composite ones. For instance, *Addition* and *Contraction* can be composed to define a *Replace* operation. Such an operation would be useful if, e.g., we acquire the information that *Michael* (an employee of NASA) is responsible for *MastCam-34 Assembly* instead of the *NASA Laboratory*. This means that the *NASA Laboratory* should be replaced by *Michael*. Figure 13 illustrates a possible definition of replacement as a composition of an addition and a contraction. Another version of replace could be formed by composing Add and Disassociate. One could similarly define more composite operations, but this exercise is beyond the scope of this work.

7 Experimental evaluation

7.1 Design of the experiments

The objective of the experimental evaluation is to understand the tradeoff between storage space, and the performance of queries and updates for different storage repositories policies. We will compare experimentally the following storage scenarios:

- Storage of K; we will refer to this scenario by *SK*.
- Storage of *C(K)* with respect to rule R1; we will refer to this scenario by SCK_{R1}.
- Storage of *C(K)* with respect to the standard RDF/S rules; we will refer to this scenario by SCK_{RDF}.
- Storage of *C(K)* with respect to *both* RDF/S rules and inference rule R1; we will refer to this scenario by $SCK_{\mathrm{RDF+R1}}$.

The main objective is to compare quantitatively the policies that could be followed regarding our update operations, contraction and disassociation, and understand the related tradeoff: a small amount of storage space and a possible overhead in query performance by storing K; or a larger amount of storage space but a better query performance in general by storing $C(K)$. The evaluation consists of three different experiments over both real and synthetic data. These experiments are explained below.

The first experiment (described in Sect. 7.2) quantifies the storage space gain of using SK as opposed to SCK_{RDF} and $SCK_{\mathrm{RDF+R1}}$. This experiment focuses on real data only, and the inference rules considered are the RDF/S rules and our custom rule R1. Due to the small size and the low complexity of the real data considered, the query and update performance is not measured in this experiment; the evaluation of query and update performance is the subject of the second and third experiments, respectively.

The second experiment (described in Sect. 7.3) consists of two sub-experiments measuring the storage space and the query execution time respectively of the two considered storage policies SK and SCK_{R1} over synthetic data. In both sub-experiments, we focus on rule R1 only; note that rule R2 applies on similar graph morphologies (so its evaluation would give similar results), whereas rule R3 is much simpler as it is not relying on hierarchies.

In the third experiment (described in Sect. 7.4), the time of maintenance and update operations for the two policies, SK and SCK_{R1}, is measured, and consists of the time required for computing and storing transitive inferences and the time for disassociating/contracting a fact from the KB, respectively. This experiment is based on rule R1 and the synthetic data used in the second experiment.

Summarizing, our evaluation consists of the following three experiments:

1. storage space evaluation over real data (Sect. 7.2)
2. storage and query execution time evaluation over synthetic data (Sect. 7.3)
3. time evaluation for computing and storing transitive inferences, and execution time evaluation for maintenance/update operations over synthetic data (Sect. 7.4)

In the figures that follow "No inference" denotes a repository containing only the initial triples (according to SK policy), while "Rule R1 inference" denotes a repository containing all inferences derived from rule R1 (according to SCK_{R1} policy).

7.1.1 Experimental settings

We performed our experiments on a Pentium 4 CPU at 2.55GHz and 2GB RAM, running Linux Ubuntu 11.10. The chosen triple store was Virtuoso's open source version 6.1.5[4]. Based on the available memory we chose to change some parameters of Virtuoso to minimize swapping. Our experiments were based on real-world data and realistic synthetic data. Specifically, our real dataset was extracted from a repository used in the 3D COFORM project (whose data concern 3D capture/acquisition), while the synthetic data were generated using *PowerGen* [60]: a RDFS schema generator that considers the features exhibited in real semantic web schemata.

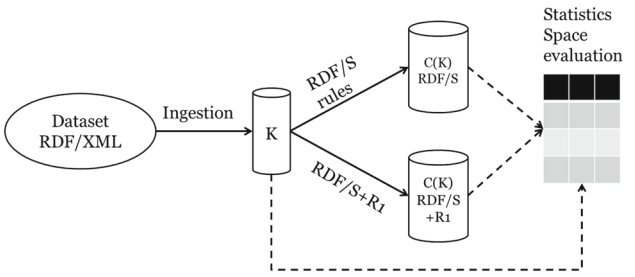

Fig. 14 Experimental design on real data

7.1.2 Input datasets

Real data Our real dataset was extracted from the most completed metadata repository available as part of the 3D COFORM project[5] in FORTH-ICS. The specific dataset was the result of a significant cooperation between institutes and museums in the context of 3D COFORM, and was collected and created manually. The model CIDOC CRM is used for encoding the metadata and the provenance of the related artefacts.

Thus, in general, it has low complexity and does not involve the whole set of classes and properties of CIDOC CRM. It consists of 150,434 triples in total; the triples involving *P9 forms part of* and *P14 carried out by* properties (which are the most critical for experiments) were 8,937 and 0, respectively.

Using the triples of this dataset, we created three different repositories. The first stored just the explicit facts (K). The second stored K along with all the inferences coming from the standard RDF/S rules (as supported by Virtuoso), i.e., SCK_{RDF}. The third stored K along with all the inferences from both the standard RDF/S rules and rule R1, i.e., $SCK_{\mathrm{RDF+R1}}$; for the latter, we defined *P9 forms part of* as a *transitive* relation, and let Virtuoso handle the transitive computation. These three repositories are shown in Fig. 14 and were used for the experiment described in Sect. 7.2.

[4] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSDownload.

[5] http://www.3d-coform.eu/.

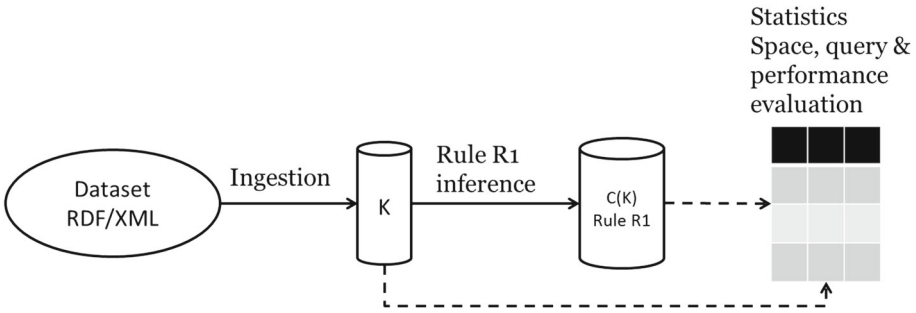

Fig. 15 Experimental design on synthetic data

Synthetic data To test the scalability of our approach and algorithms in large datasets, we synthesized data involving only the *P9 forms part of* and *P14 carried out by* properties. The goal was the evaluation of space and query performance (Sect. 7.3), and the time performance of maintenance and update operations (Sect. 7.4) with respect to our rule R1.

We decided to evaluate the effect of two parameters on synthetic data, namely, the number of the ingested triples and the depth of the activity hierarchies. Thus, having in mind these two parameters we synthesized hierarchies of the most complex form, i.e., directed acyclic graphs (DAGs[6]). To this end, we used *PowerGen* [60], a tool for generating random RDF/S schemata with DAG subsumption hierarchies considering the features exhibited by graphs of real existing semantic web schemata. The tool supports various parameters like the maximum depth and the number of classes. Using *Powergen*, we generated multiple schemata of 1,000 classes. To test our approach, we post-processed the resulting schemata, by replacing the IsA relation *RDFS:SubClassOf* with *P9 forms part of*; this has the additional effect that the multitude of *PowerGen* parameters dealing with tuning the morphology of IsA hierarchies can essentially be used to tune the morphology of *P9 forms part of* hierarchies, a critical property for our experiments (per rule R1).

The resulting repositories contained datasets of multiple disjoint unions of graphs reaching 200K, 400K, 600K, 800K (K denotes thousands) and 1M (M denotes million) *P9 forms part of* triples. For each dataset we distributed actors randomly to the activities; the number of actors considered was equal to 1 % of the *P9 forms part of* triples. The number of activities and actors did not change for different graph depths.

We stored these datasets in Virtuoso in different repositories for each dataset and applied rule R1. The inferred triples of each repository were stored in other repositories (which correspond to the SCK policy). For instance, we had two repositories for the 200K dataset: one storing *K* (i.e., *SK* policy) and one storing *C(K)* (i.e., SCK_{R1} policy). Those repositories were used for the experiments conducted in Sects. 7.3 and 7.4. Figure 15 demonstrates this process for each repository.

7.2 Space evaluation results on real data

For this experiment, we ingested our real dataset in Virtuoso. After the RDF/S inference the number of *P14 carried out by* triples increased from 0 to 129. The reason for that increase is that even though the initial data did not consist of triples having as predicate the *P14 carried out by* property, there were triples having as predicate's subproperties *P14 carried out by* (note that according to the RDF/S rules, all instances of a subproperty are also instances of its superproperty). The total number of triples (implicit and explicit) in the repository was 310,919.

Regarding the *P9 forms part of* relation, after the RDF/S inference the number of 8,937 did not change (see Fig. 16), since *P9 forms part of* is a super property defined in CIDOC CRM, so there are not other superproperties to be inferred. Subsequently, the total triples after applying rule R1 to the dataset was 327,452 and the triples regarding *P9 forms part of* and *P14 carried out by* properties were 24,444 and 717, respectively.

The number of *P9 forms part of* triples in this case has been increased since we take into account the transitivity of this relation. As a result, we observe that there was an

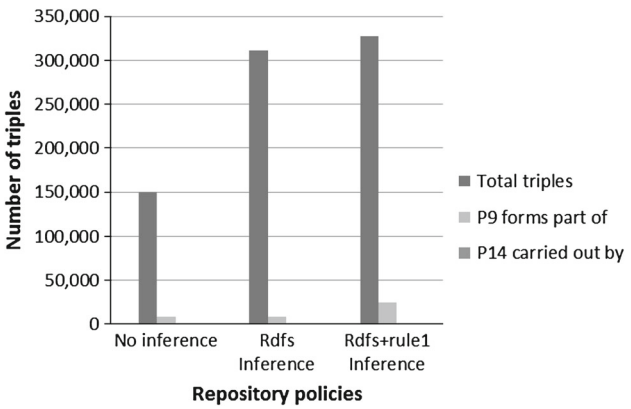

Fig. 16 Space evaluation on real data

[6] http://en.wikipedia.org/wiki/Directed_acyclic_graph.

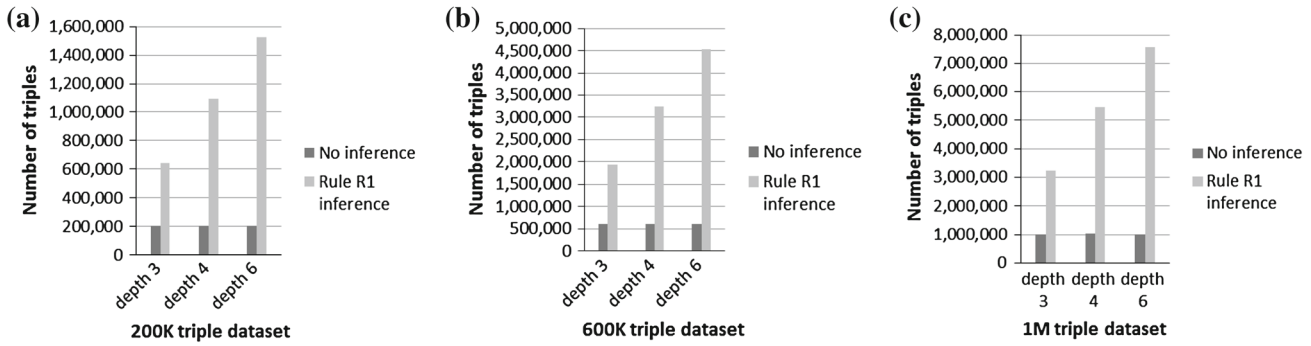

Fig. 17 Space evaluation on synthetic data **a** 200 K triple dataset, **b** 600 K triple dataset, **c** 1 M triple dataset

increase of 106 and 118% in the number of total triples from the repository containing K to the ones of the RDF/S inference and both RDF/S and rule R1, respectively. The increase percentages in the policies of storing the inferences with respect to the rule R1 and/or the RDF/S rules indicate that our approach is more beneficial considering the storage space requirements.

7.3 Space and query time evaluation results on synthetic data

7.3.1 Storage Space Requirements

In this experiment, we compare the storage space required for the storage policies SK and SCK_{R1}, and study how these requirements are affected by the maximum depth of DAGs and the number of triples. We ingested our synthetic datasets and stored the transitive and inferred triples derived by applying rule R1. In this case we do not consider any RDF/S inference. Because of this fact the data have been synthesized by replacing the IsA relation *RDFS:SubClassOf* with *P9 forms part of* (see also Sect. 7.1.2 describing this process).

In the figures below we show the results for the 200K (Fig. 17a), 600K (Fig. 17b) and 1M (Fig. 17c) triples in the respective maximum depths 3, 4 and 6. The total triples after the inference of rule R1 and for the 1M datasets was 3,229,902, 5,478,634 and 7,560,778 for 3, 4 and 6 depths, respectively.

We observe that the maximum depth of the DAGs is the main cause for the difference in size of the two policies. In particular, this difference increases as the maximum depth increases (see Fig. 17). For instance, for the 1M triples we have an increase of 218, 435 and 648 % in the number of triples for SCK_{R1}, for depths 3, 4 and 6, respectively. Finally, we notice that the storage requirements are not affected by the increase of the number of triples. For example for the dataset sizes 200K, 600K and 1M, the increase is the same and is approximately 435 %.

7.3.2 Query Performance

We also experimented with the performance evaluation of queries involving the transitive *P9 forms part of* relation. Specifically, we used the query *"retrieve all associated actors of a particular activity"* aiming at returning also the actors of its superactivities (i.e., taking into account the application of rule R1). This query is required for answering questions of the form "WHO: the persons or organizations playing role in the event" [26]. Since such questions are very often, the aforementioned query is appropriate for using it for measuring the overhead in query answering.

We evaluated this query on our synthetic datasets for all the activities in the hierarchies and their respective depth, taking the average. Every time we changed a dataset we cleared the cached buffers of the operating system to have reliable results.

Figure 18a–c shows the query time in milliseconds (ms) for the datasets of 200 K, 600 K and 1 M triples and the depths 3, 4 and 6, respectively. We observe that for the SCK_{R1} storage policy, the query time is almost the same for different dataset sizes and depths, i.e., 2 ms. This is explained by the fact that all triples are stored, so there is not any additional computation such as the transitive closure to be evaluated at query time.

In comparison, for the policy SK the query time performance, for maximum depth 6 and all datasets, has an increase reaching almost at 3.5ms (see Fig. 19). Therefore, the time required for SK is increased by almost 70–75 % over the respective one for SCK_{R1}. This is expected since the transitivity of the *P9 forms part of* relation has to be computed at query time; a computation which is not needed for SCK_{R1}. This overhead is acceptable compared to the storage space overhead, which is, on average, 433 % (see the previous experiment).

Finally, Fig. 19 shows an overview of the query time for all depths. We observe a slight performance decrease as a result of the increasing depth for almost all datasets. Therefore, as

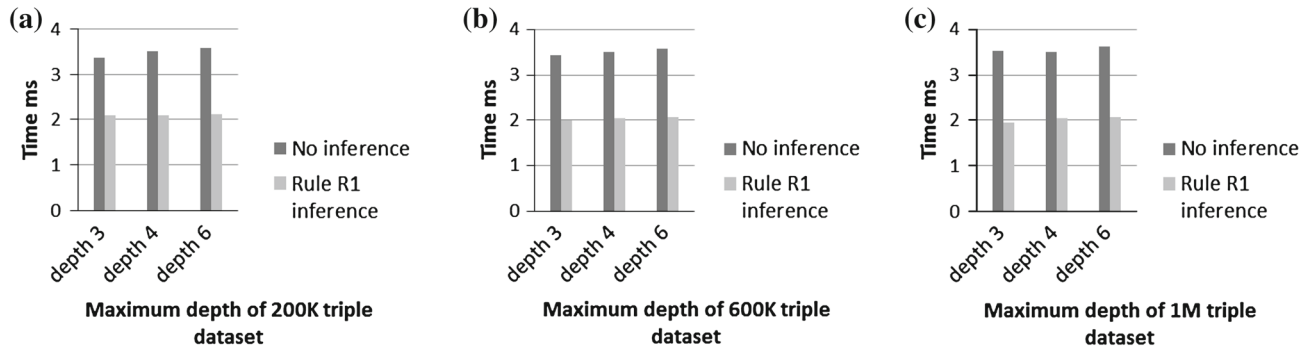

Fig. 18 Query time evaluation on synthetic data **a** 200 K triple dataset, **b** 600 K triple dataset **c** 1 M triple dataset

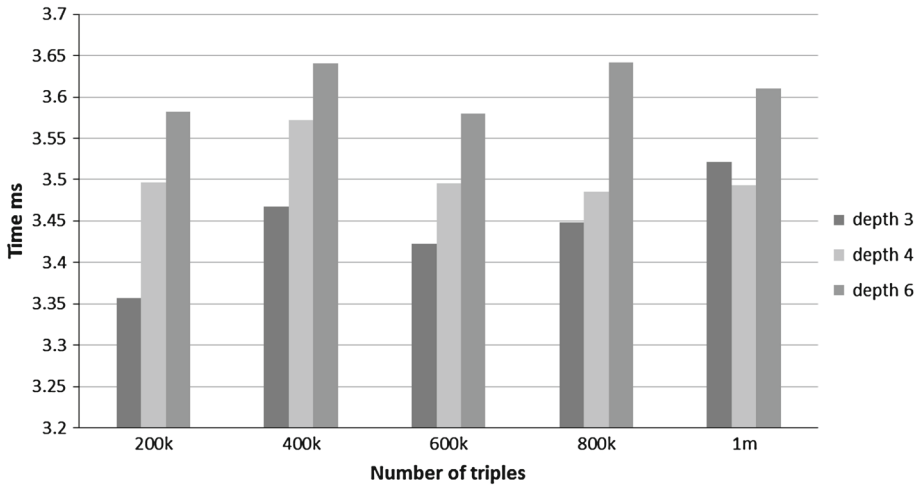

Fig. 19 Query performance for all datasets

the hierarchy becomes deeper, the number of the inferred *P9 forms part of* and *P14 carried out by* triples increases as well. Another observation that can be made is that, on average, the time needed to evaluate this query for all the activities is independent of the number of triples stored into the repository. This can be explained by Virtuoso's scalability feature [27].

7.4 Time evaluation results of maintenance and update operations on synthetic data

7.4.1 Performance of maintenance operations

One issue concerning SCK_{R1} is the time required for computing and storing the inferred relations such as those of the transitive closure (i.e., *C(K)*) of *P9 forms part of* .

We performed the discussed task for the *P9 forms part of* relation and for each synthetic dataset, to quantify this overhead and realize how the maximum depth affects the execution time for this task. Since the memory usage of this task was increased and there was some usage of the swap file, we chose to clear the swap file before starting the task for a new dataset.

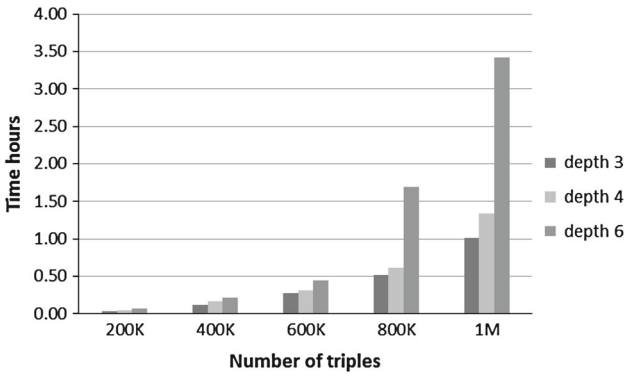

Fig. 20 Time for computation and storing the transitive closure of all datasets

Figure 20 shows the time (in hours) required to compute and store the transitive closure (SCK_{R1}) for all the triple datasets and the various depths. We observe that both the depth and the number of triples affect the time needed for the discussed task. We should emphasize the increase of time in correlation with the increase of dataset's maximum depth (see Fig. 20). Indicatively, the time for the 1 M dataset and maximum depth 6 was almost 3.5 h.

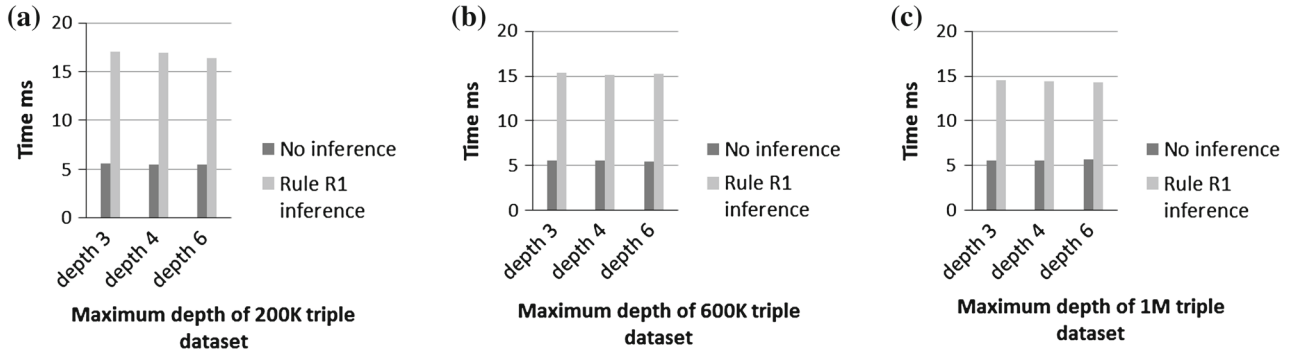

Fig. 21 Execution time evaluation of disassociation **a** 200 K triple dataset, **b** 600 K triple dataset, **c** 1 M triple dataset

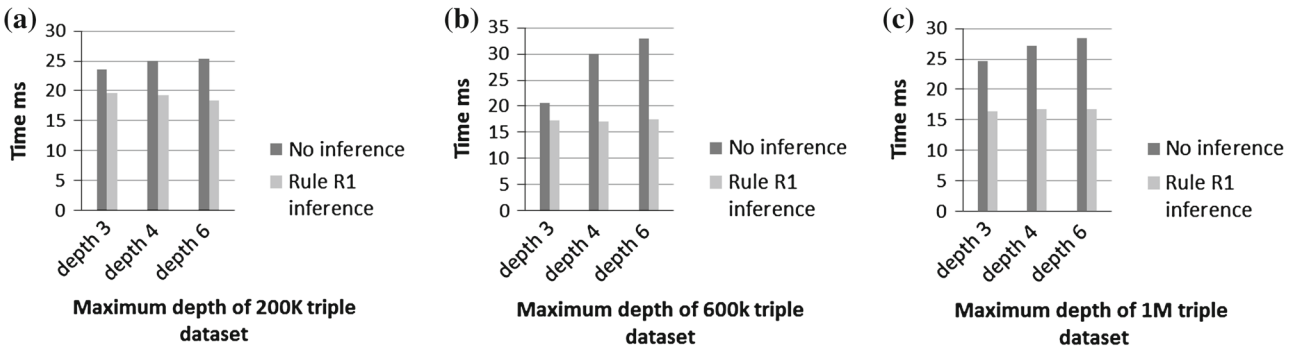

Fig. 22 Execution time evaluation of contraction **a** 200 K triple dataset, **b** 600 K triple dataset, **c** 1 M triple dataset

As a result, the *SK* policy has one more advantage (compared to SCK_{R1}), namely the lack of any maintenance overhead which, as Fig. 20 shows, is significant in the case of SCK_{R1}.

7.4.2 Disassociation time performance

In this experiment, we measure the time required for the actor disassociation update operator. We run this operation for all the actors and depths in our synthetic datasets, and report on the average time needed for the disassociation operation.

Figure 21a–c shows the results for the respective datasets and depths. We observe that the operation in the repository following *SK* requires significantly less time compared to SCK_{R1}. More specifically, for *SK* the average required time over all depths and datasets is approximately 33 % (i.e., 5ms) of the corresponding time needed for SCK_{R1} (i.e., 15 ms). The reason for this increase is mainly the larger number of triples that have to be deleted in the case of SCK_{R1} as opposed to *SK*.

7.4.3 Contraction time performance

Lastly, in this experiment we measure the time required for applying the actor contraction operator. As with the actor disassociation, we run this operation for all the actors and depths in our synthetic datasets, and took the average measurements.

The operation of contraction is more complicated than disassociation. Except from the call to a disassociation operation which is performed in the end (cf. Algorithm 2 in Sect. 6), the operation requires the computation of maximal activities and the addition of triples to preserve implicit knowledge (for *SK*).

Figure 22a–c shows the results for the respective datasets and depths. As the graphs show, in this case, the time required for contracting actors in SCK_{R1} is 67 % (i.e., approximately 17,5ms) of the time required in *SK* (i.e., approximately 26ms). This might seem strange in the light of the disassociation results, given that contraction under the SCK_{R1} policy needs to delete more triples than contraction under *SK* (as in disassociation). Thus, this observation suggests that the search and addition of explicit triples along with disassociation, which is executed at the end of the algorithm, are quite costly operations for *SK*. The former task involves the addition of new triples (which are already stored in SCK_{R1}) and the latter, as already discussed, the inference of some transitive triples at query time (which again is not needed in SCK_{R1}).

The role of disassociation (which is part of contraction) can be realized by the fact that the number of contraction

operation in SCK_{R1} is on average close to the case of disassociation for SCK_{R1}, i.e., approximately 16–17ms. Thus, the time required is mostly for the operation of disassociation in order for contraction to be completed.

7.5 Synopsis of all experimental results

Our experimental findings showed that our considered policy *SK* is very beneficial regarding the reduction of storage space requirements of provenance information, with no significant overhead in the query time performance (refer to Sects. 7.2 and 7.3). Both for the real and synthetic data the increase percentages in storage space were over 100 and 200 %, respectively (see Figs. 16 and 17), compared to SCK_{RDF+R1} and SCK_{R1}; whereas the overhead in query execution was approximately 70–75 % (see Fig. 19).

The disadvantage of SCK_{R1} is also realized by the fact that computing and storing the transitive inferences, which is a KB maintenance operation necessary in SCK_{R1} but irrelevant for *SK*, require a significant amount of time (see Fig. 20).

Lastly, regarding the update operations disassociation and contraction, the former was executed faster in *SK*, whose execution time was the 35 % of the respective one in SCK_{R1} (see Fig. 21); whereas the latter achieved better execution time in SCK_{R1}, which was 33 % better than the one in *SK* (see Fig. 22). It should be noted that for all experimental measurements involving the synthetic data the maximum depth of graph hierarchies had a quite influential role on the storage space requirements and the query performance.

Even though 1M triples could be considered a relatively small dataset, we believe that new experiments over a bigger dataset will not contradict our initial assumptions and will result to the same previously discussed conclusions (in terms of the relative reduction in the storage space requirements and the relative overhead at the query time performance). In this respect, the important feature is the structure, i.e., the form of the *RDFS:SubClassOf* hierarchy in which the nodes of the graphs were organized, rather than the size of the data. This is the reason why we have chosen to use real-world data and synthetic data that reflect the morphological features of real schemata. Consequently, the only bias introduced in the evaluation datasets is the bias imposed by the structure of a typical real-world dataset.

8 On applicability

Here we discuss some of the hypotheses used in this study.

The general assumption endorsed by this study is provenance information recorded by empirical evidence [46]. We may distinguish three epistemological situations:

1. The facts can reliably and completely be registered by a monitoring system, such as a workflow shell.
2. There are facts which users need to input manually to the monitoring system and may not be willing to do so.
3. Facts come from different monitoring systems or uncontrolled human input.

The initial provenance information is assumed to be given by the curator (or produced automatically by the system) at ingestion time. Our algorithms in Sect. 6 are focusing on provenance transformations, i.e., how one can deal with changes in the provenance information. Handling changes in the data is not in the scope of this work. Any effects of data changes on provenance are also assumed to be given to the system by the curator. For example, if the data regarding a certain measurement are deleted for some reason, then all its provenance information should be deleted as well, so the curator should explicitly execute a disassociation operation to delete such provenance information.

From a semantic point of view, the considered inference rules are based on the semantics of the involved concepts/properties, as described in the scope notes of CIDOC CRM. As such they are correct, as long as the representation of the provenance data is done according to the intended semantics of CIDOC.

Alternative rules could also be reasonable, e.g., a rule stating that a person carrying out a process carries out at least one of its sub-processes or that a person carrying out a process, also carried out all of its super-processes. That of course would give a different semantics to the "carries out" property which would be incompatible with the current semantics provided by CIDOC CRM. More specifically, in the second case, the semantics of the property "carry out" implies more a presence than a responsibility of the actors to the respective activities and it might be replaced by a more appropriate property (e.g., "was present at").

From a practical point of view, each rule causes the derivation of additional knowledge but a very large set of rules might introduce other difficulties such as further dependencies among the rules affecting the update operations (e.g., when an addition or deletion causes cascading violations of other rules, which result in more side effects for each operation [28]) and result to a more complex reasoning system.

When dealing with data that involve inference rules, there are two approaches to deal with implicit information: either computing the full closure at ingestion time or computing (part of) the closure at query time. Both approaches have their merits and drawbacks and there are commercial reasoners that follow both the first (e.g., OWLIM[7]) and the second (e.g., Virtuoso[8]) approach. Our approach is more valuable

[7] http://www.ontotext.com/owlim.

[8] http://virtuoso.openlinksw.com/.

over dynamic data, because doing changes on the full closure is generally more expensive than doing it on the ingested information. In addition, it saves space, which may be important in applications where storage space is more expensive than processing power [e.g., when the data must be sent over a slow network, such as the case with the Mars Rover (see Sects. 2 and 3) that sends data to Earth]. A further advantage of this method is that people can more easily understand and perform changes on the ingested dataset, than on the inferred one.

The curators could use the results of this paper to evaluate the above benefits and the overhead in querying and updating performance (see Sect. 7), and determine whether the proposed inference approach is suitable for their setting. In this regard, our rules, and the involved classes and properties, could be customized according that setting.

9 Related work

9.1 Knowledge evolution in RDF/S

The research field of *ontology evolution* [30] deals with updating knowledge in ontologies; a detailed survey of the field appears in [29]. However, the ontology evolution does not consider custom inference rules and does not examine provenance.

Some works (e.g., [56]) address the problem using ontology editors. However, it has been argued [56] that the naive set-theoretical application of changes that take place in most editors is insufficient, because, first, it introduces a huge manual burden to the ontology engineer, and second, it does not consider the standard inference semantics of RDF/S and other ontological languages.

In response to this need, works like [11,30,36,49,58] have proposed and implemented change semantics that consider the standard inference rules of RDF/S and determine, for each type of change request, the side effects necessary to properly execute said change taking into account the inference semantics. However, these works do not consider custom inference rules and do not discriminate between coherence and foundational semantics for their change operations (they only consider foundational semantics). In certain cases, some flexibility is provided to independently customize the semantics of some of the operations [30]. Similarly, in [42], one can explicitly define the semantics of change operators.

Some works deal with all change operations in a generic manner. For example, [43] proposes a declarative approach which can handle all possible changes on the data part of an RDF/S KB. Under this approach, operations (and their effects) are generically modelled, avoiding the need to define a specific operation for each type of change request. A similar, and more generic, approach which can handle both schema and data changes in a generic manner appears in [37]. These works consider only the standard inference rules of RDF/S (but [43] can be extended to support also custom inference rules) and only the coherence semantics (i.e., there is no support for the foundational semantics).

Finally, we should mention [34] which elaborated on the deletion of triples (including inferred ones) assuming the standard inference rules of RDF/S for both schema and instance update focusing on the "erase" operation. This work also considers generic operations, but only for the case of removing information. The considered semantics is coherence semantics (only), and the authors describe how one can compute all the "optimal" plans for executing such an erase operation (contraction in our terminology), without resorting to specific-per-operation update plans.

9.2 Relational database views

The problem discussed in this work could be related to the problem of *view updating*, i.e., define a view that contains the results of inference and then use that view also for updating.

In fact, dynamic inference rules have also been proposed in [41] in the form of non-materialized views implementing the transitive closures. In comparison to our work, the above works (i.e., [15,23,41,44,45]) do not examine updates.

Some of the first works on view updates are [10,20,22]. [22] proposes updating only when there are no side effects or under certain conditions, while the authors in [10] note that there could be more than one update policies and the choice depends on the specific application. The non-determinism of updating has also been studied by [17,24] in the context of deductive databases. In general, similar to our work, [17,24] take advantage of backward chaining searching for the justification of existing facts. Other works that follow a deterministic way of updating are [39,40,61], which propose storing both the positive and negative facts in the database. However, contrary to our intention that approach would increase the amount of information that has to be stored.

Finally we should stress that in our work the assumed framework is that of semantic web technologies since RDF triplestores are increasingly used in digital libraries. Nevertheless, even if we ignore the technological context of this work, one could not use a standard relational database. Expressing our inference rules as relational views requires adopting recursive SQL which is not supported by many DBMSs and to the best of our knowledge none supports assistance for updates through such views.

Inheritance in object-oriented databases

Provenance propagation could be compared with the notion of *inheritance* used in object-oriented databases [4,9,18,52,62]. Although, in our case, we have "part-of" instead of "IsA" relations and in some cases (e.g., for rule

R3) cannot be assumed because the rules are not applied on hierarchies of objects.

9.3 Other perspectives

Propagation of provenance during ingestion versus provenance propagation at query time

The reasoning forms that we consider in this work aim at the dynamic completion (deduction) of facts from original input by resolving transitive closures and propagating the property instances at *query time*, rather than at ingestion time. This is complementary to reasoning on *"data provenance"*, which traces causal dependencies of individual elements of datasets between input and output. For instance, and for the context of relational databases, [6] proposes algorithms to find the "core" minimal provenance of tuples in the results of equivalent queries.

We should also mention the works of [15,23,44,45]: the inference rules defined in these works are focused on the derivation of further causal dependencies between processes and artefacts and not on the propagation of features or properties among entities.

The work in [33] also uses rules to recompute provenance and determine the derivation of facts based on the provenance that exists in update exchanges related to systems for data sharing. Finally, inference rules with annotations are exploited in [14] for scalable reasoning on web data. Even though these annotations are indicators of data provenance, they do not directly model it.

Compact methods for storing provenance

The problem of efficient storage of provenance information has been extensively recognized in literature and different methods have been presented for reducing space storage requirements of provenance information. For instance, in [35] workflow DAGs are transformed into interval tree structures to encode provenance in a compact manner.

Similar to our notion of property propagation, [19] proposes provenance to inheritance methods assuming a tree-form model, but DAGs are not considered. Moreover, [35] and [19] are applied in information already stored, whereas in our case, all the inferred knowledge would have to be derived, stored and then minimized.

Analogous techniques have been proposed in [7,8]; however, none of these works elaborate on any update operation. In [8], a model more general than CIDOC CRM is presented; that work also considers DAGs and uses several inference rules for collapsing provenance traces. Anand et al. [7] mainly focus on dependency transitive relations presenting a high-level query language and reducing in parallel the storage required to represent provenance information.

Implicit versus tacit knowledge

Our approach is based on the distinction between *explicit* and *implicit* knowledge. Another distinction that has appeared in the literature distinguishes between *implicit* and *tacit* knowledge [50,54]. Tacit knowledge is the kind of knowledge gained from perception, cognition, intuition, experience and observation, e.g., the knowledge of riding a bike. In this regard, *tacit knowledge* is distinguished from explicit, because of the fact that it can be acquired through experience ("know-how"). On the other hand, our considered implicit knowledge is based solely on facts ("know-what") and is logically derived from the explicit one.

10 Concluding remarks

In this paper, we argued for the need for provenance-based inference aiming at the dynamic completion (deduction) of provenance information from the original input to reduce the storage space requirements. The inference rules considered are complementary to the ones proposed by other works that infer provenance dependencies between data, and they are mainly based on the propagation of attributes in hierarchical structures of data. This dynamic propagation may also propagate possible errors in the KB, resulting from the initial ingested data input. However, they can be easily corrected since they are only attributed to the original (explicit) data input and thus the search space for their identification is reduced; instead of searching the whole KB for errors, an examination of the ingested facts is sufficient.

The application of inference rules introduces difficulties with respect to the evolution of knowledge; we elaborated on these difficulties and described how we can address this problem. We identified two ways to deal with deletions in this context, based on the philosophical stance against explicit (ingested) knowledge and implicit (inferred) one (foundational and coherence semantics). In this regard, we elaborated on specific algorithms for these operations respecting the application of our custom rules. This presents us with two novelties, since (a) current works do not consider both approaches and (b) they take into account only the RDF/S inference rules but not any user-defined custom inference rules.

Although we confined ourselves to three specific inference rules, the general ideas behind our work (including the discrimination between foundational and coherence semantics of deletion) can be applied to other models and/or sets of inference rules.

We conducted experiments on real and synthetic data comparing different policies related to (a) storing all the inferences from the rules (i.e., $C(K)$) and (b) storing only the initial ingested facts (i.e., K, which is our proposed approach), that could be adopted by a metadata environment. The comparison was made in relation to the time performance of queries, the time performance of maintenance and update operations, and the storage space requirements for each

policy. Our results showed that the option of storing only the explicit facts (under the *SK* policy) provides significant gains with respect to storage space reduction. Moreover, the overhead in the query and update time performance was not huge, compared to the corresponding performance when storing both the explicit and implicit information (under the SCK_{R1} policy). Thus, we proved the hypothesis that our approach (*SK* policy) is advantageous.

A next step would be the examination of a larger set of inference rules, that could be interdependent, and the respective update operations. Our plan is to study the problem in a more generic manner, to deal with change operations without having to resort to specific, per-operation update plans, in the spirit of [37]. Finally, further research would be performing an extended evaluation with a larger set of queries and estimating the percentage of complete versus incomplete provenance information before and after the rules.

Acknowledgments Work done in the context of the of the following European projects: APARSEN (Alliance Permanent Access to the Records of Science in Europe Network, FP7 Network of Excellence, project number: 269977, duration: 2011–2014), DIACHRON (Managing the Evolution and Preservation of the Data Web, FP7 IP, project number 601043, duration: 2013–2016), 3D-COFORM IST IP (Tools and Expertise for 3D Collection Formation, project number: 231809, duration: 2008–2012) and PlanetData (FP7 Network of Excellence, project number: 257641, duration: 2010–2014).

Appendix A: Set of update operations

Since we focus on three inference rules, we should define operations for satisfying update requests related to these rules. The signatures of the required change operations are listed below (Table 1):

Table 1 Update operations

Signature	Rel. Rule(s)
AssociateActorToActivity (p:Actor, a:Activity)	R1
DisassociateActorFromActivity (p:Actor, a:Activity)	
ContractActorFromActivity (p:Actor, a:Activity)	
AssociateSubActivityToActivity (suba:Activity, a:Activity)	R1
DisassociateSubActivityFromActivity (suba:Activity, a:Activity)	
CreateActor (p:InstanceName)	R1
DeleteActor (p:Actor)	
CreateActivity (a:InstanceName)	R1, R2
DeleteActivity (a:Activity)	
AssociateActivityToMMObject (a:Activity, o:ManMadeObject)	R2
DisassociateActivityFromMMObject (a:Activity, o:ManMadeObject)	
ContractActivityFromMMObject (a:Activity, o:ManMadeObject)	
AssociatePartToMMObject (subo:ManMadeObject, o:ManMadeObject)	R2
DisassociatePartFromMMObject (subo:ManMadeObject, o:ManMadeObject)	
CreateMMObject (o:InstanceName)	R2
DeleteMMObject (o:ManMadeObject)	
AssociatePMMThingToEvent (ph:PhysicalManMadeThing, e:Event)	R3
DisassociatePMMThingFromEvent (ph:PhysicalManMadeThing, e:Event)	
AssociateIObjectToPMMThing (io:InformationObject, ph:PhysicalManMadeThing)	R3
DisassociateIObjectFromPMMThing (io:InformationObject, ph:PhysicalManMadeThing)	
AssociateIObjectToEvent (io:InformationObject, e:Event)	R3
DisassociateIObjectFromEvent (io:InformationObject, e:Event)	
CreateEvent (e:InstanceName)	R3
DeleteEvent (e:Event)	
CreateCarrier (ph:InstanceName)	R3
DeleteCarrier (ph:PhysicalManMadeThing)	
CreateInformationObject (io:InstanceName)	R3
DeleteInformationObject (io:InformationObject)	

Appendix B: Algorithms of update operations

Below, we list the algorithms of our set of update operations which were presented previously in Appendix A.

Algorithm 1 AssociateActorToActivity (p:Actor, a:Activity)

1: **if** an explicit *P14 carried out by* link does not exist between *a* and *p* **then**
2: Add an explicit *P14 carried out by* link between *a* and *p*
3: **end if**

Algorithm 2 DisassociateActorFromActivity (p:Actor, a:Activity)

1: **if** an explicit *P14 carried out by* link exists between *a* and *p* **then**
2: Remove the requested *P14 carried out by* link between *a* and *p*
3: **end if**
4: **for** each superactivity:superAct of *a* related to *p* via the *P14 carried out by* link **do**
5: Remove possible explicit *P14 carried out by* link between superAct and *p*
6: **end for**

Algorithm 3 ContractActorFromActivity (p:Actor, a:Activity)

```
if an explicit P14 carried out by link exists between a or a superactivity of a and p then
    for each direct subactivity:subAct of a do
        Execute AssociateActorToActivity (p, subAct)
    end for
end if
if an explicit P14 carried out by link exists between a and p then
    Remove the requested P14 carried out by link between a and p
end if
for each maximal superactivity:supAct of a related to p via the P14 carried out by link do
    for each subactivity:subAct of supAct do
        if subAct is not superactivity or subactivity of a then
            Add subAct to collection: Col
        end if
    end for
end for
Execute DisassociateActorFromActivity (p, a)
for each maximal activity:act in Col do
    Execute AssociateActorToActivity (p, act)
end for
```

Algorithm 4 AssociateSubActivityToActivity (suba: Activity, a:Activity)

```
if an explicit P16 was used for link does not exist between suba and a then
    Add an explicit P16 was used for link between suba and a
end if
```

Algorithm 5 DisassociateSubActivityFromActivity (suba: Activity, a:Activity)

```
if an explicit P16 was used for link exists between suba and a then
    Remove the requested P16 was used for link between suba and a
end if
```

Algorithm 6 CreateActor (p:InstanceName)

```
if an E39 Actor class instance with name p does not exist then
    Create an instance of the class E39 Actor with name p
end if
```

Algorithm 7 DeleteActor (p:Actor)

```
for each activity:a related to p via the P14 carried out by link do
    Remove possible explicit P14 carried out by link between a and p
end for
Remove the requested E39 Actor class instance p
```

Algorithm 8 CreateActivity (a:InstanceName)

```
if an E7 Activity class instance with name a does not exist then
    Create an instance of the class E7 Activity with name a
end if
```

Algorithm 9 DeleteActivity (a:Activity)

```
for each superactivity:superAct of a do
    Execute DisassociateSubActivityFromActivity (superAct, a)
end for
for each performer:p related to a via the P14 carried out by link do
    Remove possible explicit P14 carried out by link between a and p
end for
for each man-made object:o related to a via the "P9 was used for" link do
    Remove possible explicit P16 was used for link between o and a
end for
Remove the requested E7 Activity instance a
```

Algorithm 10 AssociateActivityToMMObject (a:Activity, o:ManMadeObject)

```
if an explicit P16 was used for link does not exist between o and a then
    Add an explicit P16 was used for link between o and a
end if
```

Algorithm 11 DisassociateActivityFromMMObject (a:Activity, o:ManMadeObject)

```
if an explicit P16 was used for link exists between o and a then
    Remove the requested P16 was used for link between o and p
end if
for each superPart:superP of o related to a via the P16 was used for link do
    Remove possible explicit P16 was used for link between superP and a
end for
```

Algorithm 12 ContractActivityFromMMObject (a:Activity, o:ManMadeObject)

```
if an explicit P16 was used for link exists between o or a superpart of o and a then
    for each direct subPart of o do
        Execute AssociateActivityToMMObject (a, subPart)
    end for
end if
if an explicit P16 was used for link exists between o and a then
    Remove the requested P16 was used for link between o and a
end if
for each maximal superPart:superP of o related to a via the P16 was used for link do
    for each subPart:subP of superP do
        if subP is not superPart or subPart of o then
            Add subP to collection: Col
        end if
    end for
end for
Execute DisassociateActivityFromMMObject (a, o)
for each maximal part in Col do
    Execute AssociateActivityToMMObject (a, part)
end for
```

Algorithm 13 AssociatePartToMMObject (subo:ManMadeObject, o:ManMadeObject)

1: **if** an explicit *P46 is composed of* link does not exist between subo and *o* **then**
2: Add an explicit *P46 is composed of* link between subo and *o*
3: **end if**

Algorithm 14 DisassociatePartFromMMObject (subo: ManMadeObject, o:ManMadeObject)

1: **if** an explicit *P46 is composed of* link exists between subo and *o* **then**
2: Remove the requested *P46 is composed of* link between subo and *o*
3: **end if**

Algorithm 15 CreateMMObject (o:InstanceName)

1: **if** an E22 ManMadeObject class instance with name *o* does not exist **then**
2: Create an instance of the class E22 ManMadeObject with name *o*
3: **end if**

Algorithm 16 DeleteMMObject (o:ManMadeObject)

1: **for** each superPart of *o* **do**
2: Execute DisassociatePartFromMMObject (o, superPart)
3: **end for**
4: **for** each activity:*a* related to *o* via the *P16 was used for* link **do**
5: Remove possible explicit *P16 was used for* link between *a* and *o*
6: **end for**
7: **for** each man-made object:mmo related to *o* via an IsA link **do**
8: Remove possible explicit IsA link between mmo and *o*
9: **end for**
10: Remove the requested E22 instance *o*

Algorithm 17 AssociatePMMThingToEvent (ph:PhysicalManMadeThing, e:Event)

1: **if** an explicit *P12 was present at* link does not exist between ph and e **then**
2: Add an explicit *P12 was present at* link between ph and e
3: **end if**

Algorithm 18 DisassociatePMMThingFromEvent (ph: PhysicalManMadeThing, e:Event)

1: **if** a *P12 was present at* link exists between ph and e **then**
2: Remove the requested *P12 was present at* link between ph and e
3: **end if**

Algorithm 19 AssociateIObjectToPMMThing (io: InformationObject, ph:PhysicalManMadeThing)

1: **if** an explicit *P128 carries* link does not exist between io and ph **then**
2: Add an explicit *P128 carries* link between io and ph
3: **end if**

Algorithm 20 DisassociateIObjectFromPMMThing (io: InformationObject, ph:PhysicalManMadeThing)

1: **if** an explicit *P128 carries* link exists between io and ph **then**
2: Remove the requested *P128 carries* link between io and ph
3: **end if**
4: **for** each physical man-made thing(carrier): ph related to io via the *P128 carries* link **do**
5: Execute DisassociatePMMThingFromEvent (ph, e)
6: **end for**
7: Execute DisassociateIObjectFromEvent (io, e)

Algorithm 21 AssociateIObjectToEvent (io:InformationObject, e:Event)

1: **if** an explicit *P12 was present at* link does not exist between io and e **then**
2: Add an explicit *P12 was present at*" link between io and e
3: **end if**

Algorithm 22 DisassociateIObjectFromEvent (io: InformationObject, e:Event)

1: **if** an explicit *P12 was present at* link exists between io and e **then**
2: Remove the requested *P12 was present at* link between io and e
3: **end if**
4: **for** each physical man-made thing(carrier): ph related to io via the *P128 carries* link **do**
5: Execute DisassociatePMMThingFromEvent (ph, e)
6: **end for**

Algorithm 23 CreateEvent (e:InstanceName)

1: **if** an E5 Event class instance with name e does not exist **then**
2: Create an instance of the class E5 Event with name e
3: **end if**

Algorithm 24 DeleteEvent (e:Event)

1: **for** each information object:io related to e via the *P12 was present at* link **do**
2: Remove possible explicit *P12 was present at* link between io and e
3: **end for**
4: **for** each physical man-made thing(carrier):ph related to e via the *P12 was present at* link **do**
5: DisassociatePMMThingFromEvent (ph, e)
6: **end for**
7: Remove the requested E5 Event class instance e

Algorithm 25 CreateCarrier (pmm: InstanceName)

1: **if** an E24 Physical Man-Made Thing class instance with name pmm does not exist **then**
2: Create an instance of the class E24 Physical Man-Made Thing with name pmm
3: **end if**

Algorithm 26 DeleteCarrier (pmm: PhysicalManMadeThing)

1: **for** each information object:io related to pmm via the *P128 carries* link **do**
2: DisassociateIObjectFromPMMThing (io, pmm)
3: **end for**
4: **for** each event e related to pmm via the *P12 was present at* **do** link
5: DisassociatePMMThingFromEvent (pmm, e)
6: **end for**
7: Remove the requested E24 Physical Man-Made Thing class instance pmm

Algorithm 27 CreateInformationObject (io:InstanceName)

1: **if** an E73 Information object class instance with name io does not exist **then**
2: Create an instance of the class E73 Information Object with name io
3: **end if**

Algorithm 28 DeleteInformationObject (io:InformationObject)

1: **for** each physical man-made thing(carrier): ph related to io via the *P128 carries* link **do**
2: Remove the *P128 carries* link between ph and io
3: **end for**
4: **for** each event:e related to io via the *P12 was present at* link **do**
5: Remove possible explicit *P12 was present at* link between io and e
6: **end for**
7: Remove the requested E73 Information Object class instance io

References

1. Definition of the cidoc conceptual reference model. http://www.cidoc-crm.org/docs/cidoc_crm_version_5.0.4.pdf
2. The Dublin Core Metadata Initiative. http://dublincore.org/
3. Riding the wave - How Europe can gain from the rising tide of scientific data (2010). http://cordis.europa.eu/fp7/ict/e-infrastructure/docs/hlg-sdi-report.pdf
4. Albano, A., Cardelli, L., Orsini, R.: Galileo: a strongly-typed, interactive conceptual language. ACM Trans. Database Syst. **10**(2), 230–260 (1985)
5. Aldeco-Pérez, R., Moreau, L.: Information accountability supported by a provenance-based compliance framework. In: UK e-Science All Hands Meeting, vol. 1 (2009)
6. Amsterdamer, Y., Deutch, D., Milo, T., Tannen, V.: On provenance minimization. In: Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. PODS '11, pp. 141–152. ACM, New York, NY, USA (2011)
7. Anand, M.K., Bowers, S., Ludäscherr, B.: Techniques for efficiently querying scientific workflow provenance graphs. In: Proceedings of the 13th International Conference on extending database technology, EDBT '10, pp. 287–298. ACM (2010)
8. Anand, M.K., Bowers, S., McPhillips, T., Ludäscher, B.: Efficient provenance storage over nested data collections. In: Proceedings of the 12th International Conference on extending database technology: advances in database technology. EDBT '09, pp. 958–969. ACM, New York, NY, USA (2009)
9. Atkinson, M., DeWitt, D., Maier, D., Bancilhon, F., Dittrich, K., Zdonik, S.: Building an object-oriented database system. chap. The object-oriented database system manifesto, pp. 1–20. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1992)
10. Bancilhon, F., Spyratos, N.: Update semantics of relational views. ACM Trans. Database Syst. **6**(4), 557–575 (1981)
11. Bechhofer, S., Horrocks, I., Goble, C.A., Stevens, R.: OilEd: A reasonable ontology editor for the semantic web. In: Proceedings of the Joint German/Austrian Conference on AI: advances in artificial intelligence. KI '01, pp. 396–408. Springer, London, UK (2001)
12. Bizer, C., Heath, T., Berners-Lee, T.: Linked data—the story so far. Int. J. Semant. Web Inf. Systems (IJSWIS) **5**(3), 1–22 (2009)
13. Boley, H.: Relationships between logic programming and RDF. In: Revised Papers from the PRICAI 2000 Workshop Reader. Four Workshops held at PRICAI 2000 on Advances in Artificial Intelligence, pp. 201–218. Springer, London, UK (2001)
14. Bonatti, P.A., Hogan, A., Polleres, A., Sauro, L.: Robust and scalable linked data reasoning incorporating provenance and trust annotations. Web Semant. Sci. Serv. Agents World Wide Web **9**(2), 165–201 (2011)
15. Bowers, S., McPhillips, T., Ludäscher, B.: Declarative rules for inferring fine-grained data provenance from scientific workflow execution traces. In: Proceedings of the 4th international conference on Provenance and Annotation of Data and Processes, IPAW'12, pp. 82–96 (2012)
16. Brickley, D., Guha, R.: Resource description framework (RDF) schema specification (2004). http://www.w3.org/TR/rdf-schema/
17. Bry, F.: Logic programming. chap. Intensional updates: abduction via deduction, pp. 561–575. MIT Press, Cambridge, MA, USA (1990)
18. Carey, M.J., DeWitt, D.J.: A data model and query language for exodus. In: Proceedings of the 1988 ACM SIGMOD international conference on Management of data. SIGMOD '88, pp. 413–423. ACM, New York, NY, USA (1988)
19. Chapman, A.P., Jagadish, H.V., Ramanan, P.: Efficient provenance storage. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. SIGMOD '08, pp. 993–1006. ACM, New York, NY, USA (2008)
20. Cosmadakis, S.S., Papadimitriou, C.H.: Updates of relational views. J. ACM **31**(4), 742–760 (1984)
21. Dalal, M.: Investigations Into a Theory of Knowledge Base Revision: Preliminary Report. In: Rosenbloom, P., Szolovits, P. (eds.) Proceedings of the Seventh National Conference on Artificial Intelligence, vol. 2, pp. 475–479. AAAI Press, Menlo Park, California (1988)
22. Dayal, U., Bernstein, P.A.: On the correct translation of update operations on relational views. ACM Trans. Database Syst. **7**(3), 381–416 (1982)
23. De Nies, T.: Constraints of the prov data model (2013). http://www.w3.org/TR/prov-constraints/
24. Decker, H.: Drawing updates from derivations. In: Proceedings of the third international conference on database theory on Database theory. ICDT '90, pp. 437–451. Springer-Verlag New York Inc, New York, NY, USA (1990)
25. Doerr, M.: The CIDOC conceptual reference module: an ontological approach to semantic interoperability of metadata. AI Mag. **24**(3), 75–92 (2003)
26. Doerr, M., Theodoridou, M.: CRMdig: a generic digital provenance model for scientific observation. In: Proceedings of TaPP'11: 3rd, USENIX Workshop on the Theory and Practice of Provenance (2011)
27. Erling, O., Mikhailov, I.: SPARQL and Scalable Inference on Demand (2009). http://virtuoso.openlinksw.com/whitepapers/SPARQL%20and%20Scalable%20Inference%20on%20Demand.pdf
28. Flouris, G., Konstantinidis, G., Antoniou, G., Christophides, V.: Formal foundations for RDF/S KB evolution. Knowledge and Information Systems pp. 1–39 (2012)

29. Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., Antoniou, G.: Ontology change: classification and survey. Knowl. Eng. Rev. **23**(02), 117–152 (2008)
30. Gabel, T., Sure, Y., Völker, J.: KAON—Ontology Management Infrastructure. SEKT informal deliverable 3.1.1.a, Institute AIFB, University of Karlsruhe (2004). http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/SEKT-D3.1.1.a.pdf
31. Gärdenfors, P.: Knowledge in Flux. Modelling the Dymanics of Epistemic States. MIT Press, Cambridge (1988)
32. Gärdenfors, P.: The dynamics of belief systems: foundations versus coherence theories. Revue Int. Philos. **44**, 24–46 (1990)
33. Green, T.J., Karvounarakis, G., Ives, Z.G., Tannen, V.: Update exchange with mappings and provenance. In: Proceedings of the 33rd international conference on Very large data bases, VLDB '07, pp. 675–686. VLDB Endowment (2007)
34. Gutierrez, C., Hurtado, C., Vaisman, A.: RDFS update: from theory to practice. In: Proceedings of the 8th extended semantic web conference on the semantic web: research and applications—Volume Part II. ESWC'11, pp. 93–107. Springer, Berlin, Heidelberg (2011)
35. Heinis, T., Alonso, G.: Efficient lineage tracking for scientific workflows. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. SIGMOD '08, pp. 1007–1018. ACM, New York, NY, USA (2008)
36. Klein, M., Noy, N.: A component-based framework for ontology evolution. In: Workshop on Ontologies and Distributed Systems at IJCAI-03, Acapulco, Mexico (2003)
37. Konstantinidis, G., Flouris, G., Antoniou, G., Christophides, V.: A Formal Approach for RDF/S Ontology Evolution. In: Proceedings of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence, pp. 70–74. IOS Press, Amsterdam, The Netherlands, The Netherlands (2008)
38. Lassila, O., Swick, R.R.: Resource description framework (RDF) model and syntax specification. W3c recommendation (1999). http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
39. Laurent, D., Luong, V.P., Spyratos, N.: Deleted tuples are useful when updating through universal scheme interfaces. In: Golshani, F. (ed.) ICDE, pp. 420–427. IEEE Computer Society (1992)
40. Laurent, D., Phan Luong, V., Spyratos, N.: Updating intensional predicates in deductive databases. In: Data Engineering, 1993. Proceedings. Ninth International Conference on, pp. 14–21 (1993)
41. Lim, C., Lu, S., Chebotko, A., Fotouhi, F.: Prospective and retrospective provenance collection in scientific workflow environments. In: IEEE SCC, pp. 449–456. IEEE Computer Society (2010)
42. Lösch, U., Rudolph, S., Vrandečić, D., Studer, R.: Tempus Fugit - Towards an Ontology Update Language. In: Proceedings of the 6th European Semantic Web Conference on the semantic web: research and applications. ESWC 2009 Heraklion, pp. 278–292. Springer, Berlin, Heidelberg (2009)
43. Magiridou, M., Sahtouris, S., Christophides, V., Koubarakis, M.: RUL: a declarative update language for RDF. In: Proceedings of the 4th international conference on the semantic web. ISWC'05, pp. 506–521. Springer, Berlin, Heidelberg (2005)
44. Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E.G., den Bussche, J.V.: The open provenance model core specification (v1.1). Future Generation Comp. Syst. 27(6), 743–756 (2011)
45. Moreau, L., Missier, P.: The PROV data model and abstract syntax notation (2011). http://www.w3.org/TR/2011/WD-prov-dm-20111018/
46. Mudge, M., Malzbender, T., Chalmers, A., Scopigno, R., Davis, J., Wang, O., Gunawardane, P., Ashley, M., Doerr, M., Proenca, A., Barbosa, J.: Image-based empirical information acquisition, scientific reliability, and long-term digital preservation for the natural sciences and cultural heritage. Eurographics Association, Crete, Greece (2008). http://www.eg.org/EG/DL/conf/EG2008/tutorials/T2.pdf
47. NASA: Science Instrument Details. http://mars.jpl.nasa.gov/msl/mission/instruments/
48. Neumann, T., Weikum, G.: x-RDF-3X: fast querying, high update rates, and consistency for RDF databases. Proc. VLDB Endow. **3**(1–2), 256–263 (2010)
49. Noy, N., Fergerson, R., Musen, M.: The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility. In: Proceedings of the 12th European Workshop on knowledge acquisition, modeling and management, EKAW '00, pp. 17–32. Springer, Berlin (2000)
50. Polanyi, M.: The Tacit Dimension. Doubleday, Garden City, NY (1966)
51. Salza, S., Guercio, M., Grossi, M., Pröll, S., Strubulis, C., Tzitzikas, Y., Doerr, M., Flouris, G.: D24.1 Report on authenticity and plan for interoperable authenticity evaluation system. Tech. rep. (2012). http://www.alliancepermanentaccess.org/wp-content/uploads/downloads/2012/04/APARSEN-REP-D24_1-01-2_3.pdf
52. Schewe, K., Thalheim, B., Wetzel, I.: Foundations of object oriented database concepts. Tech. rep., Hamburg, Germany, Germany (1992). http://www.ncstrl.org:8900/ncstrl/servlet/search?formname=detail&id=oai%3Ancstrlh%3Auhamburg_cs%3Ancstrl.uhamburg_cs%2F%2FB-157
53. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. SIGMOD Rec. **34**(3), 31–36 (2005)
54. Smith, M.K.: Michael Polanyi and tacit knowledge. The encyclopedia of informal education (2003). www.infed.org/thinkers/polanyi.htm
55. Sosa, E.: The raft and the pyramid: coherence versus foundations in the theory of knowledge. Midwest Stud. Philos. **5**(1), 3–26 (1980)
56. Stojanovic, L., Motik, B.: OntologyEvolution within Ontology Editors. In: EKAW'02/EONWorkshop, pp. 53–62 (2002)
57. Strubulis, C., Tzitzikas, Y., Doerr, M., Flouris, G.: Evolution of workflow provenance information in the presence of custom inference rules. In: 3rd International Workshop on the role of Semantic Web in Provenance Management (SWPM'12), Heraklion, Crete (2012)
58. Sure, Y., Angele, J., Staab, S.: OntoEdit: multifaceted inferencing for ontology engineering. J. Data Semant. **2800**, 2003 (2003)
59. Theodoridou, M., Tzitzikas, Y., Doerr, M., Marketakis, Y., Melessanakis, V.: Modeling and querying provenance by extending CIDOC CRM. Distrib. Parallel Databases **27**, 169–210 (2010)
60. Theoharis, Y., Georgakopoulos, G., Christophides, V.: PoweRGen: a power-law based generator of RDFS schemas. Inf. Systems **37**(4), 306–319 (2012)
61. Vrain, C., Laurent, D.: Updates, induction and abduction in deductive databases. In: European Conference on Artificial Intelligence (ECAI) Workshop on Abductive and Inductive Reasoning (1996)
62. Wilkinson, K., LyngbÃęk, P., Hasan, W.: The iris architecture and implementation. IEEE Trans. Knowl. Data Eng. **2**(1), 63–75 (1990)