PrInt: A Provenance Model to Support Integration Processes

Bruno Tomazela
University of São Paulo
São Carlos, SP, Brazil
tomazela@icmc.usp.br

Carmem S. Hara
Federal University of Paraná
Curitiba, PR, Brazil
carmem@inf.ufpr.br

Ricardo R. Ciferri
Federal Univ. of São Carlos
São Carlos, SP, Brazil
ricardo@dc.ufscar.br

Cristina D. A. Ciferri
University of São Paulo
São Carlos, SP, Brazil
cdac@icmc.usp.br

ABSTRACT

In some integration applications, users are allowed to import data from heterogeneous sources, but are not allowed to update source data directly. Imported data may be inconsistent, and even when inconsistencies are detected and solved, these changes may not be propagated to the sources due to their update policies. Therefore, they continue to provide the same inconsistent data in the future until the proper authority updates them. In this paper, we propose PrInt, a model that supports user's decisions on cleaning data to be automatically reapplied in subsequent integration processes. By reproducing previous decisions, the user may focus only on new inconsistencies originated from source modified data. The reproducibility provided by PrInt is based on logging, and by incorporating data provenance in the integration process.

Categories and Subject Descriptors

H.2.5 [**Database management**]: Heterogeneous Databases

General Terms

Management

Keywords

Data integration, reapplication of integration decisions, data provenance, logging

1. INTRODUCTION

Data integration has been the focus of a lot of attention in both academia and industry [5]. At instance level, data integration aims at solving inconsistencies on data imported from heterogeneous sources, which may contain information on the same entity in the real world, but that differ on the value of their attributes. Although there are a number of approaches proposed in the literature that investigate data integration and data provenance separately, few of them consider both in the same setting [8, 3, 2, 6, 1]. In this paper, we address the use of provenance to support instance level data integration.

Data provenance is the set of metadata that allows for the identification of sources and transformations applied to data, since its creation to its current state [7, 4]. There are several advantages of incorporating data provenance into integration processes, such as the ability to provide curated data back to external sources. However, there are a number of integration applications in which updates on external sources are not allowed, for instance, due to lack of permission. In this setting, even when the integration process identifies that two pieces of information refer to the same entity in the real world, and corrects them on local copies, these updates may not be propagated to the sources. Therefore, the same erroneous data continue to be provided until the proper authority decides to update them.

Example 1. Suppose that a research center periodically generates a list of publications of its personnel by importing data from their home pages and also from digital libraries, such as DBLP and ISI Web of Knowledge. Three researchers, *John*, *Jack* and *Mary*, are co-authors of a paper, but the imported data are inconsistent, as shown in Fig. 1a. Here, a *paper* is composed of attributes *title*, *year* and *venue*, and papers are identified by the value of the attribute *title*. Each *paper* also contains a set of *authors*, each of them with attributes *name* and *citationOrder*. We consider that in the context of a *paper*, each *author* is identified by its *name*. A user integrating these sources may perform the following actions: 1. the value of paper's *title* for *John* is manually edited from '*Integrating...*' to '*Integration...*'; 2. paper's *year* is copied from *John* to *Jack*; 3. author '*Bob*' is removed from *Mary*; 4. *venue* is copied from *John* to *Jack*; and 5. author '*Mary*' from *Jack* is inserted into *John*. The remaining actions (i.e., 6 to 9) represent copies, which are similar to actions 2 and 4. After performing these actions, the imported data become consistent and are stored as three local copies, each one containing the integrated paper shown in Fig. 1b. Due to lack of permission, corrections made in local copies may not be propagated to the sources. Thus, in subsequent integration processes the user must reprocess

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.

the imported data, by taking the same decisions, in addition to those over new detected inconsistencies.

Repeating manual interventions on integration processes has two main drawbacks. First, it is error-prone, and may lead to contradicting solutions among integration processes. Second, it is time consuming, and it tends to get worse as the volume of data increases. Therefore, another advantage of incorporating data provenance into data integration processes is to allow user's decisions to be reproduced in future iterations of the process.

The few provenance-based data integration systems that have been proposed in the literature [8, 3, 2, 6, 1] do not aim at reproducing integration decisions in applications where the integration process cannot update external data sources. In order to address this issue, a provenance model must guarantee that all decisions taken by the user in previous integrations processes are solved automatically in subsequent integration processes. For reproducing a sequence of operations and maintaining the same semantics, the provenance model has to consider the effect of transitive and overlapping operations, since they may mistakenly affect the final result of the integration process. Furthermore, it must be checked whether user's decisions remain valid. We say that a decision is valid if the context in which it has been made remains unchanged. That is, if the data sources continue to provide the same data used as basis for the decision in a previous integration process.

In order to achieve the aforementioned goals, we propose PrInt, a novel data provenance model that supports instance level data integration processes. The model focuses on applications in which data sources are read-only, i.e. users are allowed to import data, but are not allowed to update them. The goal of PrInt is the reproduction of user's decisions among distinct integration processes. Using our model, all actions performed by the user in Example 1 are reapplied automatically, guaranteeing the proper management of transitive and overlapping operations while performing validation. As a result, the user will only solve new inconsistencies generated from source modified data.

This paper is organized as follows. Section 2 gives an overview of the properties satisfied by the PrInt model, which are detailed in Sections 3 to 5. Section 6 concludes the paper.

2. THE PRINT MODEL

The integration scenario considered by the PrInt model is depicted in Fig. 2. In this scenario, a first integration process, denoted as $IntProc_1$, is executed by receiving as input several heterogeneous sources (Fig. 2a). It uses an integration tool to identify and solve inconsistencies. As the user manually decides how to solve the inconsistencies (Fig. 2b), these actions are mapped to operations stored in a *repository* (Fig. 2c). Consistent copies of the imported data are locally stored, since direct updates on heterogeneous sources are not allowed (Fig. 2d).

In a subsequent integration process, which we denote as $IntProc_2$, the local copies from $IntProc_1$ are replaced by up-to-date versions of the imported data. However, in $IntProc_2$, the sources may or may not continue to provide the same inconsistent data (Fig. 2e). Thus, before reapplying the operations defined by $IntProc_1$, we first need to check whether the source have been updated. That is, we *validate* the repository of operations with respect to the sources' current version (Fig. 2f). This step is followed by the *reapplication* of all valid operations on the imported data. This process reproduces the user's integration decisions taken in $IntProc_1$ (Fig. 2g). As a result, manual intervention for solving inconsistencies among data sources in $IntProc_2$ are limited to those new inconsistencies originated from source updates since $IntProc_1$ (Fig. 2h and b).

Performing a "*validation* and *reapplication*" preprocessing on imported data based on a *repository of operations* can drastically reduce the number of manual interventions in an integration process. As a result, integration becomes less error-prone and less time-consuming. To support this process, the PrInt model has been designed to satisfy two properties, as follows.

- **Consistency of the Repository**: Given a set of data sources S and a repository consisted of a sequence of operations Op, each data item in S is updated at most once by operations in Op.
- **Strict Reproduction of User's Decisions**: If a value conflict has been solved based on a set of data item values V, then the same strategy for solving the conflict is guaranteed to be taken by the user in subsequent processes if V remains unchanged.

The motivation for the first property is to guarantee that there are no two operations that determine the final value of a data item. In order to achieve consistency, relationships between operations must be considered. For instance, if in an integration process the user takes a decision that overrides a previous one, then operations related to the older decision must be managed properly. The management of relationships between operations is described in Section 4.

The second property, described in Section 5, refers to the idea that if a decision has been made based on a set of data values V, we can only apply the same strategy if these values remain unchanged. For instance, suppose that two data sources provide distinct values, v_1 and v_2, for the same data item, and the user chooses v_1 over v_2. Once one of them, say v_2 is modified to v_3, it is not clear whether the decision of choosing v_1 over v_3 is correct. In this case, we do not reapply the corresponding operations so that inconsistencies are not introduced in local copies without the user's consent.

Before describing the PrInt's properties, we provide in Section 3 details on operations supported by our model.

3. STORAGE OF INTEGRATION DECISIONS AS OPERATIONS

External data sources may keep information on distinct data formats. After importing these source data into PrInt, entities are represented as objects, which are composed of attributes and subobjects. We consider that each object can be uniquely identified by the value of a subset of its attributes, called key attributes.

In PrInt user's integration decisions are mapped to a sequence of operations, which are stored in a *repository*. We consider four operations: *insert*, *remove*, *edit*, and *copy*. *Insert* and *remove* are operations on objects, while *edit* and *copy* are attribute-level operations. Observe that these operations do not directly reflect user actions for integrating data as the ones described in Example 1. That is, *user*

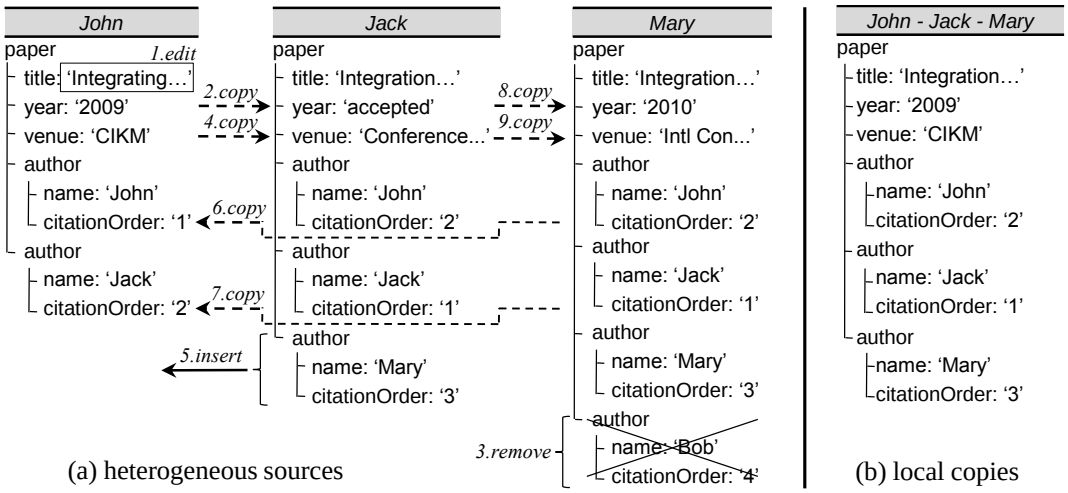

Figure 1: Inconsistent sources and actions performed by the user to integrate them.

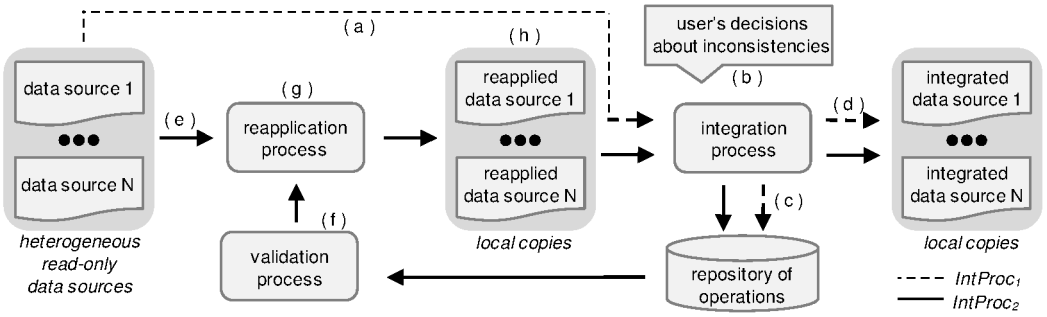

Figure 2: The PrInt model's architecture.

actions or *decisions* can be expressed in a high-level language, and be mapped to a *sequence* of basic *operations* in the repository. For instance, consider action 5, which inserts author '*Mary*' in *John*'s data, based on the values provided from *Jack*. The insert action is mapped to a sequence of operations consisted of an *insert* operation to create a new *author* subobject for *John*, followed by a *copy* operation from *Jack* to *John* on attribute *citationOrder*. Similarly, action 3 for removing author '*Bob*' from *Mary* consists of an *edit* operation for setting '*null*' to the non-key attribute *citationOrder*, followed by a *remove* operation. On the other hand, edit and copy actions are mapped to edit and copy operations, respectively.

A repository R is an array of records, where each record contains information on a basic operation. The array order reflects the temporal execution order of the operations. Each record in R is composed of the following attributes: (i) *id*: integer that identifies an operation in R ($id \geq 1$); (ii) *op*: the type of the operation, cp, ed, in, rm denoting *copy*, *edit*, *insert* and *remove*, respectively; (iii) *origin*: source that provides the correct value; (iv) *target*: source that is the target of the operation; (v) *objKey*: values of key attributes of the object; (vi) *objAtt*: attribute name involved in *op*; (vii) *originValue*: *origin*'s attribute value; (viii) *targetValue*: original *target*'s attribute value before the execution of *op*. Given a record a in R, we denote by $a.att$ the value of attribute att of a.

4. CONSISTENCY OF THE REPOSITORY

There are two types of relationships between operations in a repository that may affect its consistency: transitive and overlapping operations, which are described in Section 4.1. In Section 4.2, we define a strategy, called the *redo* policy, for detecting and managing inconsistencies among operations.

4.1 Relationships between Operations

When an operation b uses the result of another operation a, b is transitive to a. Transitive operations can be direct ($a \rightarrow b$) or indirect ($a \rightarrow_n b$), and capture the idea of propagating operations results to subsequent ones.

We also consider operations that overwrite previous results, which we denote as overlapping operations and distinguish between origin ($a \leftarrow_o b$) and target ($a \leftarrow_t b$) overlapping. When $a \leftarrow_o b$ or $a \leftarrow_t b$, operation a is said to be the *overlapped operation* and operation b is said to be the *overlapping operation*. Intuitively, $a \leftarrow_t b$ if both operations writes on the same data item, and $a \leftarrow_o b$ if operation a uses the value of a data item that is later overwritten by operation b.

Given the notions of transitive and overlapping operations, a *repository is consistent* if transitive operations do not create cycles and there are no overlapped operations. Transitive operations may propagate a value back to its origin, creating a cycle, which is not desirable because it could generate an infinite number of operations to be stored in the repository. This problem is avoided in PrInt by not allow-

ing copy operations to have the same value on its *originValue* and on its *targetValue*, i.e., in a given copy operation a, $a.originValue$ must be different from $a.targetValue$. Therefore, cycles of transitive operations never occur in PrInt.

4.2 The Redo Policy

Inconsistencies among operations are derived from overlapping operations, and are propagated to others by transitivity. Managing the repository consistency consists of developing policies for repairing inconsistencies generated from overlapping operations. These actions take place during the integration process. In this section, we propose the *redo* policy for keeping the repository consistency. This policy is maintained by modifying and reordering operations stored in the repository.

Consider first origin overlapping. If $a \leftarrow_o b$ then operation a uses the value $a.originValue$ for updating an attribute value of another object on data source $a.target$; $a.originValue$ is later overwritten by operation b with a different value $b.originValue$. In this case, we infer that the user's decision for applying operation a is based on the fact that she chooses the value provided by $a.origin$ over that provided by $a.target$. In order to maintain this decision, given that $a.originValue$ is updated by operation b with $b.originValue$, this new value should also be propagated to $a.target$ (i.e., $b \rightarrow a$ becomes true). Therefore, the *redo* policy for solving this inconsistency is to modify a by changing $a.originValue$ to $b.originValue$, and moving a to the end of the repository. Operations that are direct or indirect transitive to a also use $a.originValue$. Thus, the same strategy is applied to these operations. Operations modified by the *redo* policy must be imediatelly executed using their new $originValue$.

Now, consider target overlapping operations. Recall that if $a \leftarrow_t b$ then both operations update the same data item in some source (i.e., $a.target = b.target$). Since operation b is executed after a, in the end result the outcome of a's execution has no effect on the integration process. Thus, the *redo* policy removes a from the repository. Nevertheless, operations that are transitive to a must be adjusted. This is because given an operation c, if $a \rightarrow c$, c takes as input the value written by a. Since the final value of this data item is now given by operation b, $c.originValue$ is changed to $b.originValue$, and c is moved to the end of the repository.

5. REAPPLICATION OF OPERATIONS

The main goal of the PrInt model is to provide a means for reapplying operations that reflect the same decisions made by the user in previous integration processes. Since data sources are autonomous, they can be updated between two integration processes. Thus, we need to validate the operations before reapplying them. Validating an operation consists of verifying if both *origin* and *target* attributes still store the same value on the data item involved. Intuitively, we can only assume that the user would take the same action for solving the conflict if the data remain unchanged. Otherwise, the model would require additional user input.

Validation has the property that if an operation is *invalid*, then all its transitive operations are invalid as well. Another property is that validation is executed at attribute level. *Copy* and *edit* actions satisfy this condition, but *insert* and *remove* actions do not, as they are defined over objects. In order to perform validation, *insert* actions are mapped to *insert* and *copy* operations, and *remove* actions are mapped to *remove* and *edit* operations, as described in Section 3. Thus, an *insert* action is valid if its corresponding *insert* and *copy* operations are valid, and a *remove* action is valid if its corresponding *remove* and *edit* operations are valid.

To reapply operations, we propose VRT (*Validate and Reapply in Tandem*), a method which consists of the following steps. It first checks whether each operation in the repository is valid with respect to the *origin*. If this condition is true, then it also checks for validity with respect to the *target*. Given that the operation is valid both on *origin* and *target*, it is reapplied on *target*. After processing all operations in the repository, the integrated data sources are available to be used in the new integration process, with all previous decisions made by the user already in place.

6. CONCLUSION

In this paper, we propose PrInt, a provenance model that supports instance level data integration processes. The model focuses on systems in which the integration process is not allowed to update heterogeneous sources directly according to user's integration decisions. Therefore, it updates local copies of data sources, and keeps a consistent repository of operations for reproducing user's decisions in subsequent integration processes automatically.

We are analyzing new policies for the management of overlapping operations to support a wider range of integration scenarios. We also plan to extend PrInt to support schema level integration [5]. Another future work is to deal with concurrent schedule of operations in multiuser environments.

Acknowledgments. This work has been supported by the following Brazilian research agencies: FAPESP, CNPq, CAPES and FINEP.

7. REFERENCES

[1] D. W. Archer, L. M. L. Delcambre, and D. Maier. A framework for fine-grained data integration and curation, with provenance, in a dataspace. In *TaPP*, 2009.

[2] O. Benjelloun, A. Das Sarma, A. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *The VLDB Journal*, 17(2):243–264, 2008.

[3] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *SIGMOD*, pages 539–550, 2006.

[4] J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for computational tasks: A survey. *IEEE Computing Science & Engineering*, 10(3):11–21, 2008.

[5] A. Y. Halevy, A. Rajaraman, and J. Ordille. Data integration: The teenage years. In *VLDB*, pages 9–16, 2006.

[6] Z. G. Ives, T. J. Green, G. Karvounarakis, N. E. Taylor, V. Tannen, P. P. Talukdar, M. Jacob, and F. Pereira. The orchestra collaborative data sharing system. *SIGMOD Record*, 37(3):26–32, 2008.

[7] A. Kementsietsidis and M. Wang. Provenance query evaluation: what's so special about it? In *CIKM*, pages 681–690, New York, NY, USA, 2009. ACM.

[8] N. Shiri and A. Taghizadeh-Azari. Lineage tracing in mediator-based information integration systems. In *ISSADS*, pages 267–282, 2005.