XML Data Fusion

Frantchesco Cecchin[1], Cristina Dutra de Aguiar Ciferri[2], and Carmem Satie Hara[1]

[1] Federal University of Paraná – Curitiba, PR – Brazil
{frantchesco, carmem}@inf.ufpr.br
[2] University of São Paulo – São Carlos, SP – Brazil
cdac@icmc.usp.br

Abstract. Ensuring high quality data when collecting and integrating information from heterogeneous sources into a data warehouse is a challenging problem. In this paper, we propose a model for XML data fusion, which allows the integrator to define data cleaning rules for solving value conflicts that may have been detected during the integration process. These rules resemble decisions that are made by users when data are manually curated and, once defined, conflicts detected in subsequent integration processes that are within the context of existing rules can be automatically solved without user intervention. We also introduce a notion of fusion policy validation that prevents conflicting resolution rules to be defined. To validate our proposal, we developed XFusion, a rule-based cleaning tool that stores curated data in a integrated repository.

1 Introduction

Nowadays companies of all sizes and from different segments maintain a repository of data imported from a number of sources. The integration of imported data in a single repository provides a unified view of the available information, and also constitutes the basis for applying data analysis techniques, such as data mining and multidimensional analysis. In fact, data warehousing has emerged as an area in recognition of the value and role of information, providing integrated and high quality information targeted at decision support.

Imported data are often inconsistent. Thus, for achieving full integration, data usually goes through an iteration of integration and cleaning processes. Here, integration refers to the problem of identifying overlapping data in different sources. This problem has been the subject of extensive research on relational [9], entity-relationship [11], and XML [16] data models. Cleaning refers to the process of solving attribute value conflicts. The problem arises when two or more sources contain information on the same entity or attribute, but disagree on their values. A number of approaches have been proposed in the literature for addressing this problem, including data profiling, data mining, constraint-based, and ontology-based techniques [13, 19, 9, 17]. Recently, the process of combining multiple records representing the same real-world object into a single, consistent, and clean representation has been denoted as data fusion [3].

The majority of existing systems for data fusion considers data structured on relational format. Nevertheless, given that XML has become the standard for data exchange on the Web, it is natural to also consider this format for the integration process. In addition to the fact that currently most data sources provide their data in XML, features that make this format suitable for data exchange are also desirable for data cleaning. One of these features is its hierarchical structure, which naturally represents relationships between entities. Furthermore, as more organizations view the Web as an integral part of their communication and business, the importance of integrating XML data in data warehousing environments is becoming increasingly high [7].

Data cleaning usually requires some manual user intervention, even though this is an error-prone and time-consuming process. In this paper we propose a data fusion model for minimizing the amount of user mediation for data cleaning in integration processes. The model is based on establishing a policy, composed of a set of rules, which resemble decisions that are usually made by users when data are manually curated. Once the policy is defined, conflicts detected in subsequent integration processes that are within its context can be automatically solved without user intervention.

The data integration scenario we consider is depicted in Figure 1. The input for an `integration` tool is a set a data sources $S_1, \ldots, S_n$. This tool is responsible for identifying corresponding entities among sources, and also for detecting value conflicts. The user defines a set of rules for solving these conflicts, which are stored in a `policy` base. Rules are then applied by a `cleaning` tool. "Discarded" data values are stored in a `resolution log`, and a clean, consistent view of the data is provided to the user by the `data repository`.

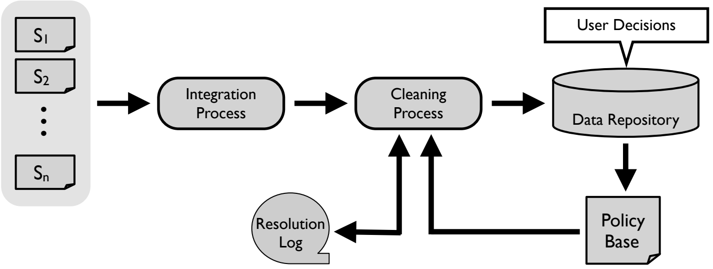
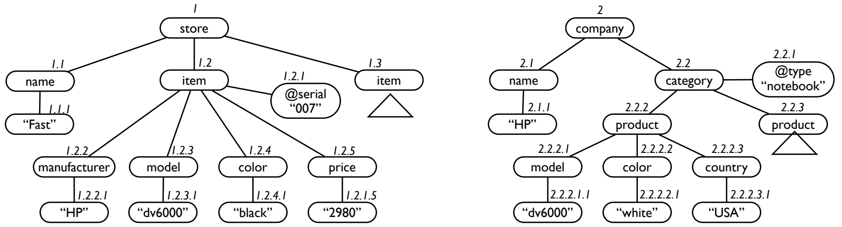


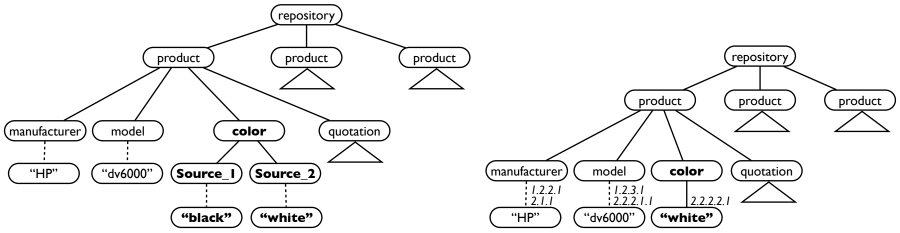
Fig. 1. Integration process with policy-based conflict resolutions for data cleaning.

Example 1. Consider two data sources providing data on `products` as depicted in Figure 2. Data from source S_1 are extracted to populate a data repository, depicted in Figure 3(a), as follows. Each `item` element is mapped to a `product`, along with its subelements `manufacturer`, `model`, and `color`. The `item`'s `price` is mapped to a child of the `product`'s `quotation`, which stores price values from different stores. For S_2, the value for `manufacturer` is extracted from `company`'s `name`, and values for the remaining subelements are given by subelements of `category/product`. We assume that `product` elements in the repository are identified by their `manufacturer` and `model`. Since both S_1 and S_2 provide data



(a) XML tree from S_1 (b) XML tree from S_2

Fig. 2. Samples of XML tree representation.



(a) Conflicting element. (b) Conflict solved.

Fig. 3. Data repository tree with temporary inconsistency [15] and after cleaning.

on products that coincide on the values of these elements, they are merged in the repository. Nevertheless, they disagree on the value of the product's `color`. Following the integration model proposed in [15], value conflicts are explicitly represented as depicted in Figure 3(a), along with their provenance, i.e. the sources that provided the conflicting data.

A cleaning strategy for solving the conflict may determine that whenever a data item provided from S_2 disagrees with any other source, we should choose S_2's value over the others, since S_2 contains data provided by the product manufacturer while other sources are resellers. As a result of applying this strategy, the data repository keeps a single consistent value for all product's subelements, as shown in Figure 3(b). In our model, a rule expressing a cleaning strategy is stored in the `policy base`, while the discarded value of product's `color` (`''black''`) is kept in the `resolution log`, which helps us consider the value in future conflict resolution processes. As an example, suppose that data from a new source S_3 is imported into the repository, and that S_3 also provides the value `''black''` for the same product. If the strategy for solving the conflict is modified for choosing the value provided by the majority of the sources, we would be unable to determine that the value for `color` in the repository should be changed to `''black''` if we did not keep S_2's value in the resolution log. Rules for solving value conflicts can be defined on different contexts of an XML tree. They may involve a single element or a set of subtrees. The ability to define contexts on subtrees may generate inconsistencies among conflict resolution rules. As an example, suppose that the rule for choosing S_2 over other data sources is to be applied on all subtrees for which the product's `manufacturer` is `HP`, and a second rule for choosing the value provided by the majority of the sources is defined on all subtrees for which the product's `model` is `dv6000`. Since both rules apply to the product in our running example, we need a notion of policy validation for avoiding such inconsistencies and for deterministically determine which rule should be applied for solving a given one.

Contributions. In this paper we make the following contributions.

- We propose a model for XML data fusion based on a set of rules for solving value conflicts in data integration processes. Cleaning rules resemble decisions commonly made by users for handling value conflicts, and minimize manual intervention for data curation.
- We define a fusion policy validation on a set of conflict resolution rules. Our fusion policy validation prevents inconsistent rules to be defined on the same element of the repository.
- We present a tool, called XFusion, that has been developed based on our model. It supports both XML data integration and cleaning processes.

Organization. The rest of the paper is organized as follows. Section 2 presents preliminary definitions used as the basis for our proposal, while Section 3 introduces our model for XML data fusion. Fusion policy validation is the subject of Section 4, and Section 5 presents the XFusion tool. Section 6 discusses related work and Section 7 concludes the paper.

2 Preliminary Definitions

Before describing our fusion model, we present a definition of XML trees, the integration model considered in this paper, and strategies for data fusion previously proposed in the literature.

2.1 XML Trees and Integration Model

An XML document is typically modeled as a node-labeled tree T, in which the set of nodes V can be of one of three types: element, attribute and text nodes. For each node n in T we define the following functions: (1) $lab(n)$ assigns a label to n if n is an element or attribute node, and a distinct label L if n is a text node; (2) $val(n)$ assigns a string to attribute and text nodes, and is undefined for element nodes; (3) $ele(n)$ and $att(n)$ define the edge relation of T: if n is an element then $ele(n)$ is a *list* of elements and text nodes in V and $att(n)$ is a *set* of attributes in V; if n is an attribute or a text node then $ele(n)$ and $att(n)$ are undefined; (4) $id(n)$ assigns a unique identifier to n, which represents the path from the root r of the tree to n. We assume that each XML tree has a distinct identifier S which denotes its source, and that $id(r) = S$. That is, the root's node identifier coincides with the XML tree source identity.

Examples of XML trees are given in Figures 2(a) and 2(b), where each node n is represented with its identifier ($id(n)$), a label ($lab(n)$) if it is an element or attribute node, and a value ($val(n)$) if it is an attribute or text node. The encoding adopted by the identifier function id is called *Dewey Order* [5], which provides a global node ordering. Since in our model each data source has a distinct source identifier, which coincides with the root identifier, in Figure 2(a), $id(r) = 1$ and in Figure 2(b), $id(r) = 2$.

In this paper, we adopt the model proposed in [15] for identifying corresponding entities among data sources, and for explicitly representing value conflicts. This model assumes that the data repository has a fixed schema, and a set of XML keys [4] for identifying elements in a document. The repository is populated with data imported from data sources through a transformation language that maps source data to the repository schema. Whenever two source elements are mapped to an entity in the repository that coincide on their key values, they are merged. The repository stores provenance information on every imported data item, and explicitly represents value conflicts detected after the merging.

Example 2. Consider again XML data sources depicted in Figures 2(a) and 2(b), and the XML tree resulting from merging them given in Figure 3(a). Following the syntax proposed in [4], the XML keys that determine how elements are merged in the repository can be defined as follows.

- $k_1 : (\epsilon, (product, \{manufacturer, model\}))$: in the context of the entire document (ϵ denotes the root), a `product` is identified by its `manufacturer` and `model` number;
- $k_2 : (product, (color, \{\}))$: within the context of any subtree rooted at a `product` node, there exists at most one `color` element; that is, it is identified by an empty set of values.

Observe that in the repository, `manufacturer` is populated with nodes reached by path $/item/manufacturer$ in S_1, and nodes reached by path $/name$ in S_2. Similarly, element `model` is populated with nodes reached by path $/item/model$ in S_1 and path $/category/product/model$ in S_2. Given that values of nodes reached by both paths in sources S_1 and S_2 coincide, they are be merged in the repository according to k_1. The resulting tree is depicted in Figure 3(a).

In the repository, leaf nodes are annotated with provenance information. These annotations are important not only to determine the origin of data, but they also allow the portion of source XML tree used to populate the data repository to be reconstructed [15].

XML keys involve path expressions. An algorithm for deciding path containment for the fragment of XPath involved in defining keys is presented in [4], while [12] and [6] investigate the problem for larger fragments of XPath. It has been shown that for some of these fragments containment can be checked in PTIME. The problem of determining intersection of XPath expressions has also been investigated in the context of query optimization [8].

2.2 Strategies for Data Fusion

There are a number of strategies proposed in the literature for solving value conflicts. We adopt a subset of strategies proposed in [2], described as follows.

Trust Your Friends. This strategy is based on a reliability criterion. The user assigns a confidence rate for each source, and a value conflict is solved by choosing the one provided by the source with the highest confidence rate.

Meet In The Middle. This is a strategy to mediate the conflict by generating a new value that is a compromise among all conflicting values, e.g., an average of all conflicting numeric values.

Cry With The Wolves. This strategy is defined for choosing the value reported by the majority of data sources.

Roll The Dice. This strategy randomly chooses one value among the conflicting ones.

Pass It On. This is a non-resolving strategy. Although in most cases the user wants a single value for each data item, for some items she may want to keep all the conflicting values in the repository. When this is the case, this choice can explicitly be made applying the *Pass It On* strategy.

In [2] these strategies are integrated to the relational model by developing functions that can be used within SQL sentences to solve inconsistencies from the resulting data set. Next section presents our model, which extends this strategy-based conflict resolution approach for XML.

3 XML Fusion Model

Our model for solving conflicts detected during the integration process is based on the definition of a fusion policy, which consists of a set of data conflict resolution rules, defined as follows.

Definition 1. *A* **conflict resolution rule** *is a pair* $\langle \sigma, \Sigma \rangle$*, where*

(1) σ *is a path expression representing the context covered by the rule;*

(2) Σ *is a non empty list of strategies for handling instance-level conflicts on nodes reached by following the context path* σ*.*

The context of a rule is defined by a path expression σ and therefore it may cover not only a single element or attribute node, but a *set* of nodes reached by following σ. Furthermore, a rule may define a *list* of strategies for solving a conflict. Thus, if the first strategy is not able to single out a value for a given data item, the following strategies are considered one by one until either the end of the list is reached or the conflict is solved. If the conflict is solved, we say that the rule *effectively solves the value conflict.*

Example 3. Consider the value conflict between a product's `color` depicted in Figure 3(a). Suppose the following rule has already been defined in the fusion policy: $\langle /product[manufacturer\ =\ \text{``}HP\text{''}]/color, [\textit{Trust Your Friends}, \textit{Cry With the Wolves}]\rangle$. It determines that whenever there is a value conflict on element `color` of `product`, and the `manufacturer` of `product` is `''HP''`, then the strategy *Trust Your Friends* should be applied, followed by *Cry With the Wolves.* Assuming that the confidence rate of S_2 is higher than S_1, strategy *Trust Your Friends* is applied and the value `''white''` from S_2 is chosen to be stored in the repository, as shown in Figure 3(b).

Observe that some strategies for solving conflicts may depend on the provenance of the data, as exemplified in the previous example by strategy *Trust Your Friends*. Data provenance should also be kept for discarded values. In our model this information is kept in a *resolution log*, which is defined as follows.

Definition 2. *A* **resolution log** *is a set of records, where each record refers to a data value v that has been discarded during the cleaning process. Given that v has been populated from an element e of a data source S, the record that refers to v contains the following attributes: (1) key values of the element or attribute with value v in the repository; (2) the discarded value v; (3) $id(e)$ in the original source S; (4) the path from the root to e in the source S; (5) the strategy $s \in \Sigma$ applied for solving the conflict.*

We need to keep the keys for the element for which a value has been discarded in order to retrieve all the discarded values for the same data item in the repository. This may be necessary for automatically reapplying a conflict resolution rule in future cleaning processes. Both the identity and the original path of the element which provides v are stored in the log for keeping provenance information. By storing the `strategy` executed to solve the conflict, we can trace back why `value` v has been discarded.

Example 4. Consider again the value conflict depicted in Figure 3(a) and the conflict resolution rule in Example 3. The record for S_1's discarded value `‘‘black’’` stored in the resolution log contains the following data: `(key: /product[manufacturer=‘‘HP’’ and model=‘‘dv6000’’]/color, value: ‘‘black’’, id: 1.2.4.1, path: /item/color, strategy: Trust Your Friends)`. Recall that from the value of *id* it is also possible to get the source identification, given that the first number of the sequence corresponds to the root element, which coincides with the source identifier.

To illustrate how the log is used in future fusion processes, consider that source S_3, as defined in Example 1, has been integrated into the repository, and that it has the same confidence rate as S_2. Given that both sources S_2 and S_3 have the same confidence rate, and that S_3 provides the value `‘‘black’’` for `color`, strategy *Trust Your Friends* is not able to solve the conflict. Then, the next strategy, *Cry With The Wolves* is applied. In this case, `‘‘black’’` is chosen to be stored in the repository, since this strategy chooses the value reported by the majority of the sources. Our model is only able to make such a decision because the log maintains S_1's discarded value for `color`.

Given the definitions of conflict resolution rules and repository log, we are now ready to define our fusion model.

Definition 3. *A* **data fusion model** $\mathcal{D}$ *is a 5-tuple* $\langle \mathcal{R}, \mathcal{T}, \mathcal{K}, \mathcal{P}, \mathcal{L} \rangle$*, where:*

(1) $\mathcal{R}$ is a set of pairs $(S, rank)$, where S is an XML data source, and rank its confidence rate; the value of rank is greater for sources with higher reliability;

(2) $\mathcal{T}$ is the data repository tree with a set of nodes V such that each leaf node $v \in V$ is annotated with a set of pairs (id_n, p), where id_n is the identifier

of a node n in a source XML tree T_S used to populate v, and p is the path in T_S from its root to n;

(3) $\mathcal{K}$ is the set of XML keys defined on $\mathcal{T}$. Every element node in $\mathcal{T}$ can be uniquely identified according to keys in $\mathcal{K}$;

(4) $\mathcal{P}$ is a set of conflict resolution rules that define strategies for solving data conflicts;

(5) $\mathcal{L}$ is the resolution log for storing data discarded during a fusion process.

An example of a data repository tree is given by the XML tree depicted in Figure 3(b). Observe that leaf nodes have been annotated with node identifiers from *Source 1* and *Source 2*. Paths traversed from the root have been omitted for simplicity.

4 Fusion Policy Validation

Recall that conflict resolution rules are defined on contexts described as path expressions. Since a path expression σ denotes a *set* of nodes in a XML tree reached by following σ, there may exist nodes that are covered by more than one rule. In order to deterministically single out a rule for solving a value conflict, we introduce a notion of *policy validity.*

In the following definition, we denote as $Nodes(r)$ the set of nodes covered by a rule r. That is, given a rule $r = (\sigma, \Sigma)$ and an XML tree T, $Nodes(r)$ is the set of nodes in T reached by following σ in T.

Definition 4. *Given two rules r_1 and r_2, we say that r_1* **is valid with respect to** *r_2 if they satisfy one of the following conditions:*

(1) $Nodes(r_1) \subset Nodes(r_2)$ or

(2) $Nodes(r_1) \supset Nodes(r_2)$ or

(3) $Nodes(r_1) \cap Nodes(r_2) = \emptyset$.

Intuitively, rules can be related either by *specialization* (Case 1) or *generalization* (Case 2), or not related at all (Case 3). Following the traditional definition of class hierarchy, Cases 1 and 2 allow rules to be defined on different levels of a tree hierarchy. That is, there may exist a general rule for solving value conflicts, but it may be overridden by rules defined for treating specific cases, that are restrict to subsets of nodes covered by the general rule, and are used in the cleaning process instead of the general rule.

Rules with intersecting coverage that are not related by specialization / generalization are not allowed. This is because there is no deterministic way of deciding which rule should be applied for solving conflicts on nodes covered by both rules. For determining the validity of a fusion policy, each conflict resolution rule should be valid with respect to all others. Checking validity involves checking both path expressions containment and intersection. Some previous work on these subjects are presented in Section 2. The model we propose in this paper is orthogonal to the path language adopted. The following definition establishes an order for applying conflicting resolution rules in a fusion policy.

Definition 5. *Let v be an element or attribute with conflicting values, and $\mathcal{P}$ a fusion policy, consisted of a set of rules that are valid with respect to each other. Rule $r_v \in \mathcal{P}$ is applied for solving the value conflict on v if:*

(1) $v \in Nodes(r_v)$ and r_v effectively solves the value conflict;

(2) there exists no rule $r_i \in \mathcal{P}$ that satisfies condition (1), such that $Nodes(r_i) \subset Nodes(r_v)$.

Example 5. Consider the data repository depicted in Figure 4, and four conflict resolution rules, defined as follows:
$r_a = (/product[manufacturer = \text{“}HP\text{”}\ and\ model = \text{“}tx1220\text{”}]/color, \Sigma_a)$
$r_b = (/product[manufacturer = \text{“}HP\text{”}]/color, \Sigma_b)$
$r_c = (/product/quotation[store = \text{“}Fast\text{”}]/price, \Sigma_c)$
$r_d = (/product[manufacturer = \text{“}Sony\text{”}]/quotation/price, \Sigma_d)$

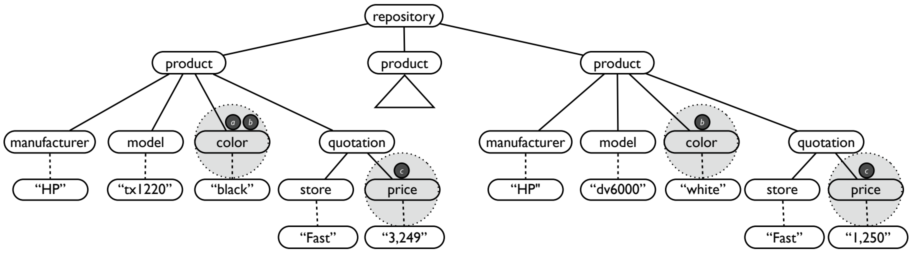


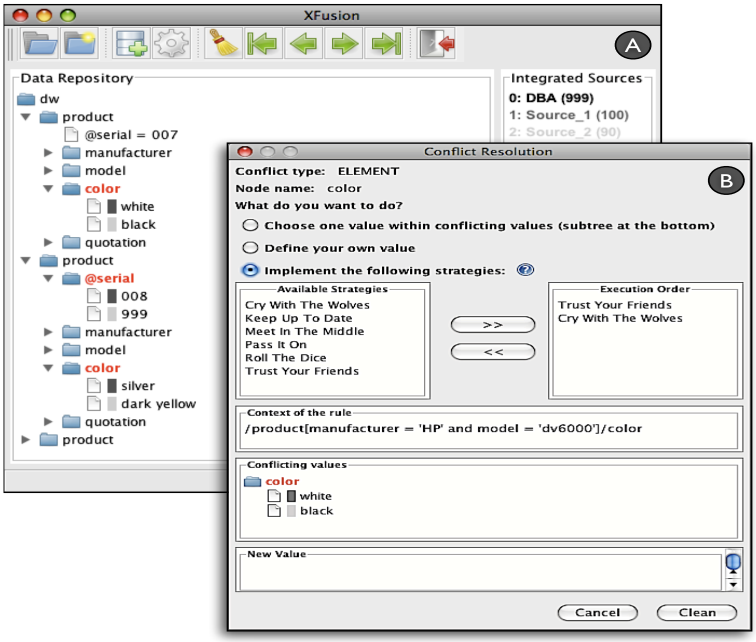
Fig. 4. Example of conflict between resolution rules.

Rules r_a and r_b are defined on `color` elements, while r_c and r_d are defined on `price`. Observe that r_a is a specialization of r_b, since $Nodes(r_a) \subset Nodes(r_b)$, and they are valid with respect to each other. On the other hand, r_c is *not* valid with respect to r_d since it is not true that for all possible XML trees $Nodes(r_c) \cap Nodes(r_d) = \emptyset$, although in the tree depicted in Figure 4 this condition holds.

5 XFusion

In order to validate the proposed XML fusion model and fusion policy validation, we have developed the XFusion tool. The tool integrates several data sources into a data repository, and presents to the user the detected value conflicts along with a set of available strategies for solving them.

Two screenshots of XFusion's interface are presented in Figure 5. Screenshot A is the main screen of the tool. It shows in the *Data Repository* panel the data repository in a tree format, and in the *Integrated Sources* panel the source's names that have been considered in the integration process. In this panel, the DBA represents the source of data items that have not been imported from external sources, but locally produced. The tool assigns a distinct color for each of the integrated sources, so that value conflicts are shown along with "colored" representation of sources that provided them (little squares that precede each value). Furthermore, screenshot A contains buttons to perform actions over the



```
dw
▼ product
    @serial = 007
  ▶ manufacturer
  ▶ model
  ▼ color
      white
      black
  ▶ quotation
▼ product
  ▼ @serial
      008
      999
  ▶ manufacturer
  ▶ model
  ▼ color
      silver
      dark yellow
  ▶ quotation
▶ product
```

0: DBA (999)
1: Source_1 (100)
2: Source_2 (90)

Conflict type: ELEMENT
Node name: color
What do you want to do?

○ Choose one value within conflicting values (subtree at the bottom)
○ Define your own value
● Implement the following strategies: ?

Available Strategies:
- Cry With The Wolves
- Keep Up To Date
- Meet In The Middle
- Pass It On
- Roll The Dice
- Trust Your Friends

Execution Order:
- Trust Your Friends
- Cry With The Wolves

Context of the rule: /product[manufacturer = 'HP' and model = 'dv6000']/color

Conflicting values:

```
color
  white
  black
```

New Value:

Fig. 5. XFusion screenshots: **A** – Main screen; **B** – Resolution screen.

data repository, such as add new source (), resolve conflicting values (), and navigate through conflicts (). To add a new data source, the user must provide a mapping that determines how source data are extracted and then inserted into the repository. She also defines the confidence rate to be assigned to the new source.

Data fusion is the main functionality of the tool based on the model proposed in this paper. When the user selects an existing conflict and clicks on the resolve button, the screen depicted in screenshot B of Figure 5 is shown. Observe that the user has three main options for solving a conflict: choose one among the conflicting values, manually insert a new value, or apply some of the available strategies, which are described in Section 2.2. Strategies are chosen by clicking on direction buttons in the middle of the screen, determining the order in which they should be considered.

Below the strategies boxes, the *Context of the rule* is presented. This path is originally set to uniquely identify the conflicting element or attribute, according to the XML keys defined on the repository. Nevertheless, the user can edit the path for applying the list of strategies on different contexts. In the current implementation, we only consider simple XPath expressions (without wildcards) with simple predicates involving elements, attributes and string values. When the user edits the context of the rule, our tool validates the rule with respect to all existing ones according to the policy validation described in Section 4. Finally, when she clicks on *Clean* button, the new rule is inserted into the policy base and its execution propagates the chosen value to the repository, and the discarded ones to the resolution log. XFusion allows the user to define rules incrementally. That is, in the first iteration a single strategy may be defined on a context path and applied. The user can then check whether the strategy has been effective for solving all conflicts within the rule's context. If not, she may decide to extend the rule by defining additional strategies to be applied.

XFusion has been implemented in Java, using the Swing graphical package. For manipulating XML documents we used JDOM. The data repository is stored on eXist-db [10], a native XML database system. XFusion implements the XML integration model proposed in [15] combined with our fusion model, showing the feasibility of our approach.

6 Related Work

Data integration and cleaning have been studied extensively by the database community [1, 3]. Most of previous works consider data on relational format, but recently it has been stressed the need for investigating the problem of solving conflicts on semi-structured data. XClean [19] is a system that allows declarative and modular specification of a cleaning process. As oppose to our approach which adopts strategies, XClean is based on operators. Systems like Potter's Wheel [18] and Fusionplex [14] are also strategy-based systems, but they allow the definition of a single cleaning strategy. To the best of our knowledge, our fusion model is the first to define a general framework for applying strategy-based techniques for solving value conflicts that maintains discarded data values in a log repository. This approach allows strategies to be applied in subsequent integration processes, and also keeps provenance information for tracing back cleaning processes. Our model also builds on previous works for determining XPath expressions containment [12, 6] and intersection [8] in order to determine fusion policy validity.

7 Conclusion

In the relational model, fusion strategies are usually defined on the context of an attribute value. The model proposed in this paper naturally extends this notion by allowing strategies to be defined on subtrees of an XML document. Our notion of policy validation also extends relational fusion policies by allowing strategies to be defined on subsets (supersets) of nodes reached by previously defined rules, by specialization (generalization). Furthermore, we guarantee that there are no two rules that can be applied on a data item, except for those related by specialization/generalization. The ability to define value conflict resolution rules on subtrees can drastically reduce the amount of user mediation on data cleaning processes, given that conflicts detected in subsequent integration processes that are within the context of existing rules can be automatically solved. The repository log plays an important role in keeping the necessary data for supporting this functionality, while also storing provenance data for tracing purposes. The model has been validated by developing XFusion, a tool based on the proposed model, which shows that it can be incorporated in a data integration and cleaning application.

Some issues that need to be further investigated include: (1) integration of the model to a technique for incrementally updating the data repository when new versions of data sources become available; (2) extensions to the fusion policy, by supporting new strategies and a declarative definition of rules application, allowing for instance, conditional execution of strategies; (3) experimental study for solving conflicts in real-world applications and for determining the cost of applying our rule-based cleaning policy.

References

1. Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. IEEE Data Eng. Bull. 29(2), 4–12 (2006)
2. Bleiholder, J., Naumann, F.: Conflict handling strategies in an integrated information system. In: Proceedings of IIWeb (2006)
3. Bleiholder, J., Naumann, F.: Data fusion. ACM Comp. Surveys 41(1), 1–41 (2008)
4. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.C.: Reasoning about keys for XML. Information Systems 28(8), 1037 – 1063 (2003)
5. Chan, L.M., Mitchell, J.S.: Introduction to the Dewey Decimal Classification. `http://www.oclc.org/dewey/versions/ddc22print/intro.pdf` (2003)
6. Genevès, P., Layaïda, N.: Deciding XPath containment with MSO. Data & Knowledge Eng. 63(1), 108 – 136 (2007)
7. Golfarelli, M., Rizzi, S., Vrdoljak, B.: Data warehouse design from XML sources. In: Proc. of DOLAP (2001)
8. Hammerschmidt, B.C., ad Volker Linnemann, M.K.: On the intersection of XPath expressions. In: Proc of IDEAS. pp. 49 – 57 (2005)
9. Lim, E.P., Srivastava, J., Prabhakar, S., Richardson, J.: Entity identification in database integration. Information Sciences 89(1) (1996)
10. Meier, W.: eXist-db open source native XML database. `http://exist.sourceforge.net/` (2000)
11. Menestrina, D., Benjelloun, O., Garcia-Molina, H.: Generic entity resolution with data confidences. In: Proc. of VLDB Work. on Clean Databases (2006)
12. Miklau, G., Suciu, D.: Containment and equivalence for a fragment of XPath. J. of the ACM 51(1), 2 45 (2004)
13. Milano, D., Scannapieco, M., Catarci, T.: Using ontologies for XML data cleaning. In: OTM Workshops. pp. 562–571 (2005)
14. Motro, A., Anokhin, P.: Fusionplex: resolution of data inconsistencies in the integration of heterogeneous information sources. Info. Fusion 7(2), 176–196 (2006)
15. do Nascimento, A.M., Hara, C.S.: A model for XML instance level integration. In: Proc. of SBBD. pp. 46–60 (2008)
16. Poggi, A., Abiteboul, S.: XML data integration with identification. In: Proc. of DBPL (2005)
17. Rahm, E., Do, H.H.: Data cleaning: Problems and current approaches. IEEE Data Eng. Bull. 23(4), 3–13 (2000)
18. Raman, V., Hellerstein, J.M.: Potter's wheel: An interactive data cleaning system. In: Proc. of VLDB. pp. 381–390 (2001)
19. Weis, M., Manolescu, I.: Declarative XML data cleaning with XClean. In: Proc. of CaiSE. pp. 96–110 (2007)