XClean in Action

A Demonstration of Declarative XML Data Cleaning

Melanie Weis
Hasso Plattner Institut Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
melanie.weis@hpi.uni-potsdam.de

Ioana Manolescu
INRIA Futurs
4 rue Jacques Monod
91893 Orsay Cedex, France
ioana.manolescu@inria.fr

ABSTRACT

We demonstrate XClean, a data cleaning system specifically geared towards cleaning XML data. XClean's approach is based on a set of cleaning operators. Users may specify cleaning programs by combining operators using the declarative XClean/PL language, which is then compiled into XQuery. We plan to show XClean in action on several scenarios based on real-world data. A graphical user interface supports users in writing XClean/PL programs and guides them through the process to obtain the clean data.

1. MOTIVATION

Data cleaning is the process of correcting anomalies in a data source, that may for instance be due to typographical errors, formatting differences, or duplicate representations of an entity. It is a crucial task in customer relationship management, data mining, and data integration. Relational data cleaning is performed in specialized frameworks [6, 10, 14], or by specialized modules in modern relational database management systems [3].

With the growing popularity of XML and the large volumes of XML data becoming available, approaches to effectively and efficiently clean XML data are needed. In developing such an approach, some of the lessons learned from the relational data cleaning experience clearly apply.

Modularity. Data cleaning processes should be *modular* in order to allow the composition of such processes from a set of smaller, interchangeable building blocks. Modularity brings several benefits. It facilitates reusing existing cleaning transformations, simplifies the process of debugging and inspecting the data transformation process, and it allows incremental development, maintenance and evolution of the cleaning process.

Declarativity. By declaratively describing the cleaning process, its logic can be decoupled from the actual processing and its implementation. This makes data cleaning processes easier to write and to debug than alternative approaches, based on imperative code. Declarative cleaning programs allow concentrating on the cleaning tasks, while delegating the storage and optimization issues to the underlying data management systems.

DBMS-backed data cleaning. Many transformations involved in data cleaning are closely related to those typically applied inside database management systems (DBMSs). Therefore, cleaning data on top of a DBMS allows taking advantage of its functionalities, including persistence, transactions etc. but also query optimization, which may speed up the cleaning.

3rd *Biennial Conference on Innovative Data Systems Research (CIDR)* January 7-10, 2007, Asilomar, California, USA.

Some XML data cleaning features make a departure from its relational counterpart, raising new challenges and opportunities:

Cleaning rich-structure data. Different XML representations of similar real-world objects may exhibit bigger differences among them than similar tuple-based ones. Thus, object properties may be multi-valued, the order and the hierarchical organization of data inside elements may vary. Furthermore, crucial information describing the way XML nodes relate to one another is encapsulated by their parent-child relationships. Thus, XML data cleaning needs to preserve and exploit these relationships, whereas relations between tuples were fully captured by the values they contained.

Cleaning XML with XQuery? Relational data cleaning's reliance on RDBMSs was limited by expressive power mismatches between the cleaning primitives and SQL. Features such as user-defined aggregate functions, transitive closure computation, nested tables etc. are either not fully supported by the language, or not well supported by existing systems. In contrast, the standard XML query language, XQuery, is Turing-complete, raising the question whether simply writing XQuery queries may not suffice for data cleaning? While this approach can be made to work, it amounts to writing fresh code for every new cleaning problem, which does not agree with our modularity requirement.

XML cleaning functions. Typical data cleaning steps, such as distance computation or duplicate detection, are quite complex and/or expensive [11, 13], and the best way to implement them may not be via XQuery. XQuery supports external functions [15], and RDBMSs feature robust execution techniques for queries using such functions [7], which can be easily adopted by XQuery processors, too. Thus, valuable, non query-like libraries useful for XML cleaning can still be exploited while cleaning XML with XQuery.

We present XClean, the first modular, declarative system for native XML data cleaning. In XClean, cleaning processes are modeled using a a set of *cleaning operators*, that can be combined in arbitrarily complex cleaning processes. The operators' specification is expressed in a high-level *operator definition language*, called XClean/PL. Writing XClean programs is supported by a graphical user interface. An XClean/PL program is *compiled* into XQuery, to be executed on top of any XQuery processor. The demo shows the expressive power and ease of use of XClean by means of several case studies.

The paper is organized as follows. We present the overall system in Sec. 2. We then describe the proposed demo scenarios in Sec. 3. Related work is discussed in Sec. 4, and Sec. 5 concludes.

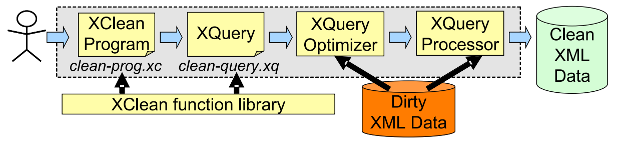


Figure 1: XClean Architecture

2. XCLEAN IN A NUTSHELL

In this section, we introduce XClean, the first declarative and modular XML data cleaning system. We outline its architecture, then describe its cleaning operators, and briefly present XClean/PL, a programming language to specify these operators.

Due to space constraints, we are very brief on these descriptions. Further details are available at [1].

2.1 XClean Architecture

The architecture of the XClean system is depicted in Fig. 1. A user specifies an XClean program in our proposed declarative XClean/PL language (see Sec. 2.3). An XClean/PL program describes a set of XClean operators, and the way their inputs and outputs are connected. The system provides a function library including commonly used functions (e.g., date formatting for scrubbing, edit distance for string similarity, transitive closure for clustering), which may be used in XClean/PL programs. The function library can be extended by user defined cleaning functions, defined as XQuery functions and implemented either in XQuery or in an external language [15].

We compile XClean/PL programs into XQuery; executing such an XQuery then outputs the clean XML data. This allows to execute the programs on top of any XQuery-enabled platform. The XQuery standard language [15] is a feature-rich language widely implemented by major DBMS vendors (such as IBM, Oracle, Microsoft etc.), and free-source projects (e.g. Saxon, BerkeleyDB/XML etc.). The interest of the XClean/PL language in itself is to provide custom syntax for cleaning-specific operators, increasing the *readability and ease of maintenance* of cleaning programs, while being significantly more *concise* than the resulting XQuery programs. The XQuery is input to an XQuery optimizer before it is executed by an XQuery processor, who outputs a clean representation of the dirty input XML data.

The focus of the demo is on the expressive power of the few XClean operators and XClean/PL. The current implementation does not feature any optimizations but the underlying XQuery engine's. We plan to use Saxon B and/or MonetDB.

2.2 Operators

XClean's cleaning operators are summarized in Tab. 1. Any XClean operator inputs and outputs collections of (nested) tuples. A tuple consists of attribute-value pairs; values may be XML nodes, atomic values, or tuple sets. We use tuples as they are convenient for modeling associations of nodes and values which must be considered jointly during cleaning, as we demonstrate on the following example.

Fig. 2 (bottom) presents a sample XML document containing three versions of the same real-world object (in this example, a movie), with their respective title, year and actor sets. The additional labels $m1$, $a1$ etc. uniquely identify an element and are used to reference them in our example. Assume that the goal of the cleaning process is: (i) obtaining one representation for each movie, including *all alternative titles*, *one year*, and *all actors (but each actor only once)*, and (ii) *restructuring each actor element* into a firstname and a lastname element. A possible result of this process is shown at the top of Fig. 2.

We now explain how this cleaning process is implemented in XClean. The process and its intermediary results can be followed from the bottom up on Fig. 2.

Operator	Goal
Candidate Selection (CS)	Select elements to be cleaned.
Scrubbing (SC)	Remove errors in text (typos, format, ...).
Enrichment (EN)	Specify data that supports cleaning.
Duplicate filtering (DF)	Filter non-duplicate element pairs.
Pairwise duplicate classification (DD)	Classify pairs of elements as duplicates, non-duplicates, ...
Duplicate clustering (DC)	Determine clusters of duplicates.
Fusion (FU)	Create unique representation of an entity.
XML view (XV)	Create XML view of clean data.

Table 1: Overview of XClean Operators

The *candidate selection (CS)* operator is used to define the elements subject to data cleaning. In our case, these are the ⟨movie⟩ and ⟨actor⟩ elements in the sample dirty XML document doc at the bottom of Fig. 2. Candidate selection operators ((1) and (2) in the figure) generate two separate tuple sets for movie and actor candidates, respectively.

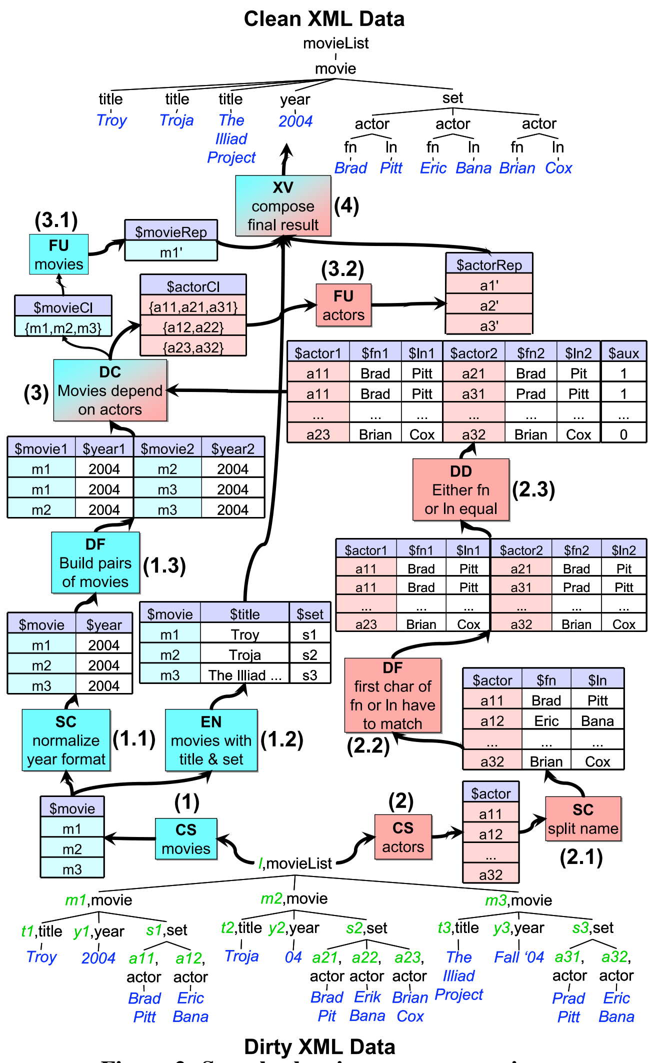
To split the texts appearing inside ⟨actor⟩ elements into first name and last name components, we apply a *scrubbing* operator (2.1). Scrubbing is reserved to simple processing of atomic values (that is, text nodes or attribute values). Similarly, to prepare for the task of choosing a single year per movie, we standardize date formats to a four-digit representation, using another scrubbing operator (1.1).

To help decide which ⟨movie⟩ elements represent the same real-world object, we annotate each movie with some extra information: its title, and set of actors. We use an *enrichment (EN)* operator to associate such extra information to cleaning candidates. In this simple example, movies were enriched with information extracted from the same document (1.2), namely their title and their actor set. In general, enrichment may add to cleaning candidates interesting data from other sources, such as, e.g., alternative titles, or directors, which in our case may be obtained from a source such as the Internet Movie Database.

A central task in data cleaning is duplicate detection, i.e. detecting multiple representations of a same real-world object. In its most general form, this process involves pairwise comparisons among the cleaning candidates. To support flexible specification of duplicate detection tasks, while allowing for their efficient implementation, XClean provides three distinct operators.

Duplicate filtering (DF) allows to prune pairs of cleaning candidates of which it can be declared with certainty that they do not represent the same real-world object. In our example, we could, for instance, specify that actors whose last names do not start with the same letter, and whose first names do not start with the same letter, either, are not duplicates (2.2). We model such pair pruning by a separate operator to provide a way for users to inject their knowledge of the application domain in the cleaning process. This leads to avoiding expensive computations (such as sophisticated distance measures, or the application of clustering procedures) whenever possible, and also has the advantage of minimizing intermediary cleaning results. Clearly, the quality of the filter influences the quality of duplicate detection, because pairs falsely filtered will never be found to be duplicate. For the pairs surviving duplicate filtering, we provide two duplicate detection approaches.

Pairwise duplicate detection (DD) considers one pair of cleaning candidates at a time, and classifies it in one class among: *duplicates*, *non-duplicates*, and possibly other classes, e.g., reflecting confidence levels such as *possible*, *unlikely* etc. In our example, we use pairwise duplicate detection to decide that two actors are duplicates if either their firstname or their lastname are equal, otherwise, they are non-duplicates, i.e., correspond to distinct real-world objects (2.3).



$movieRep
m1'

$movieCl
{m1,m2,m3}

$actorCl
{a11,a21,a31}
{a12,a22}
{a23,a32}

$actorRep
a1'
a2'
a3'

$actor1	$fn1	$ln1	$actor2	$fn2	$ln2	$aux
a11	Brad	Pitt	a21	Brad	Pit	1
a11	Brad	Pitt	a31	Prad	Pitt	1
...	...	...	...	...	...	...
a23	Brian	Cox	a32	Brian	Cox	0

$movie1	$year1	$movie2	$year2
m1	2004	m2	2004
m1	2004	m3	2004
m2	2004	m3	2004

$actor1	$fn1	$ln1	$actor2	$fn2	$ln2
a11	Brad	Pitt	a21	Brad	Pit
a11	Brad	Pitt	a31	Prad	Pitt
...	...	...	...	...	...
a23	Brian	Cox	a32	Brian	Cox

$movie	$year
m1	2004
m2	2004
m3	2004

$movie	$title	$set
m1	Troy	s1
m2	Troja	s2
m3	The Illiad ...	s3

$actor	$fn	$ln
a11	Brad	Pitt
a12	Eric	Bana
...	...	...
a32	Brian	Cox

$movie
m1
m2
m3

$actor
a11
a12
...
a32

Figure 2: Sample cleaning process overview.

Pairwise duplicate detection is simple and natural to use, however, it does not fit all application scenarios and duplicate detection algorithms. Whereas pairwise duplicate detection applies locally on one pair of (possibly enriched) candidates at a time, we define our *duplicate clustering (DC)* operator to work globally, taking as input sets of candidate pairs. Duplicate clustering can be applied to perform transitive closure over a set of detected duplicate pairs returned by a pairwise duplicate detection method [8], but it also supports the detection of duplicates using algorithms that rely on relationships between them [4]. In our example, duplicate clustering produces both actor clusters, and movie clusters (3). Their detection relies on the observation that relationships between movies and actors may be used to help cleaning *both*.

From every cluster, a unique cleaned representation is obtained using the *fusion (FU)* operator. In our example, ⟨actor⟩ fusion (3.2) must reconcile the exact spelling of their first and last names, whereas ⟨movie⟩ fusion (3.1) requires resolving conflicts in their actor sets, years, and titles.

Our last operator, called *XML view (XV)*, is used for XML restructuring operations, e.g., to put together the partial results of various cleaning operators into a cleaned document, or to align differently-structured candidates into a unique syntax. In our example, XML view encapsulates scrubbed firstnames and lastnames into elements, keeps alternative movie titles and associates fused actors with fused movies (4).

```
ENRICH $m IN $scrubbedMovies
INTO $enrichedMovies
BY $m.mCand/movie/title/text() AS $title,
   $m.mCand/movie/set/actor AS $set;
CLUSTER CLASSIFICATION USING
   $xcl:radc($actorDups, $candMovieDups)
INTO $movieClusters SCHEMA [$movieCluster],
     $actorClusters SCHEMA [$actorCluster];
```

Table 2: Sample XClean/PL clauses.

2.3 XClean Programming Language

The specification of a cleaning process can be decomposed in two parts: (i) the specific filters, distance functions, duplicate detection algorithms, clustering algorithms etc. that the user chooses; (ii) and the "surrounding" code necessary to implement the operators based on the functions (as described in the previous section), and to glue the operators among them.

Previous experience in data cleaning [6, 10, 14] demonstrates that creating or choosing the cleaning functions and algorithms requires a human expert, and cannot be automated. In contrast, the second task is repetitive, and amenable to automation. Based on this observation, we designed the XClean/PL language as follows.

An XClean/PL program is a set of clauses, each of which defines a cleaning operator. Operators input and output tuples from shared, global XClean/PL variables. Sample XClean/PL clauses appear in Tab. 2. XClean/PL keywords appear in bold font.

The top enrichment clause defines the operator labeled (1.2) in Fig. 2. The clause refers to two named tuple sets, globally visible in the XClean/PL program: $scrubbedMovies, the operator's input, and $enrichedMovies, its output. The tuple variable $m iterates over the input. The **BY** clause introduces the two enrichments: the result of each query is added as a new variable, part of the output flow.

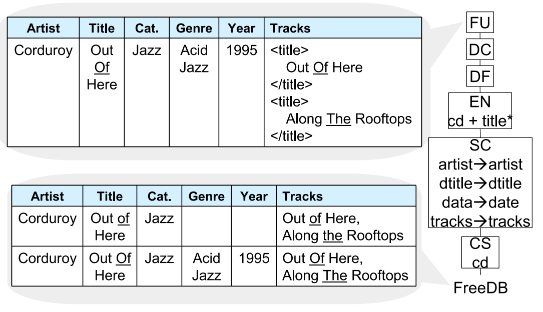
The cluster classification clause defines the operator labeled (3) in Fig. 2. The classifier function $xcl:radc denotes a relationship-aware duplicate clustering function [13], which is one among the possible classifiers to be used here. The classification function returns two sets of clusters, one containing movies and another one actors. The **INTO** keyword is used, as previously, to capture the outputs of $xcl:radc, and make them visible in the XClean program for further usage. Furthermore, this clause explicitly renames the attributes in each set of cluster's schema, through the **SCHEMA** clause.

The full description of XClean/PL's syntax, its translation to XQuery, and more detailed examples are delegated to [1].

3. DEMO SCENARIOS

The demo will focus on XClean's expressive power, and highlight the benefits of modular and declarative XML data cleaning, by means of several real-life dirty XML data sets, outlined in this section. Use case details, sample data sets, full XClean/PL programs, and their resulting XQueries, are available at [1].

FreeDB Use Case. This use case concerns CD description data from the FreeDB site (http://www.freedb.org). The cleaning process (Fig. 3) includes correcting errors in artist names (e.g., different capitalization schemes, Various Artists is also represented by V.A., Various), standardizing dates, correcting titles (e.g., the title element often includes Artist/Title), and track titles (again, capitalization). Furthermore, the text including a comma separated list of track titles is split into several track title elements. The final task is to de-duplicate CDs: if both ⟨artist⟩ and ⟨title⟩ are equal, we consider CDs to be duplicates. Clusters of duplicates are formed, and fused to a single representative for every CD. During fusion, conflicts may appear in category, genre, year, and track titles: different categories, genres, and years are concatenated, whereas sets of ⟨title⟩ elements are unified. Note that the table representation in Fig. 3 has only been used for readability, the actual data is XML.



Artist	Title	Cat.	Genre	Year	Tracks
Corduroy	Out Of Here	Jazz	Acid Jazz	1995	<title> Out Of Here </title> <title> Along The Rooftops </title>

Artist	Title	Cat.	Genre	Year	Tracks
Corduroy	Out of Here	Jazz			Out of Here, Along the Rooftops
Corduroy	Out Of Here	Jazz	Acid Jazz	1995	Out Of Here, Along The Rooftops

Figure 3: FreeDB Use Case

MOVIE Use Case. This is a data integration scenario, in which movies from two sources are first mapped to a common schema, and then de-duplicated. The source data originates from the Internet Movie Database IMDB (http://www.imdb.com) and the German Movie Repository FILMDIENST (http://film-dienst.kim-info.de/). Fig. 4 outlines the two source schema and the target schema. In IMDB titles, the possible leading "The" or "An" in a paper's title is separated in an ⟨article⟩ element. Non-trivial correspondences between source and target types are rendered by curved arrows, possibly annotated with transformation functions. For example, IMDB names are split into a firstname and a lastname, and the gender is set to "m" for actor, or to "f" for actresses. In addition to performing the transformation into the target schema, we also scrub and de-duplicate the data using the same duplicate detection algorithm used in our motivating example.

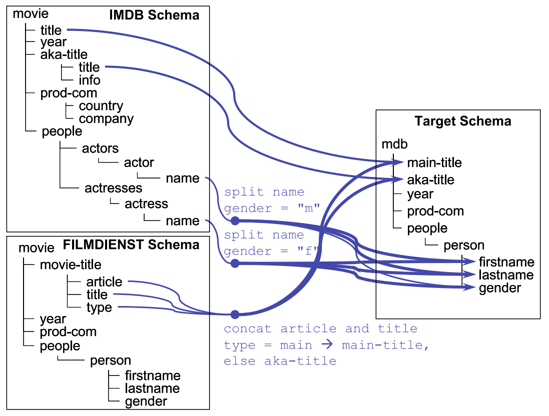


Figure 4: MOVIE Use Case

DBLP Use Case. In the well-known DBLP data source some authors with identical names (e.g., Albrecht Schmidt) share the same page, whereas some researchers' works are split across several pages due to different name spellings. The co-authors of an author are a good starting point to fix these data problems: the co-author sets of the two same-name researchers are disjoint, while the co-author sets of a single person (whose name is spelled in different ways) may overlap. A clustering algorithm working on the co-author relationship is designed to repair these problems, and applied within XClean.

CORA Use Case. The CORA bibliographic data set is frequently used to evaluate duplicate detection algorithms [4, 12]. The sample XClean process scrubs, enriches and restructures the dirty data. Using the restructured data, publications, dates, authors and venues are deduplicated, and are assigned an identifier. In this example, we again scrub dates, reusing a standard function available in the XClean function library already used in the MOVIE scenario. Also, detecting duplicates in author names is similar to detecting duplicates in actor names, so we can reuse the same pairwise duplicate detection function as in the MOVIE scenario, showing the advantage of modularity.

4. RELATED WORK

We only briefly discuss selected related work to data cleaning systems, and refer to [9] for a survey on relational data cleaning. More recent approaches approaches include AJAX [6] and Potter's Wheel [10]. XClean is conceptually close to AJAX by its operator-based approach, however the XML context lifts the expressive power barriers that confronted AJAX. In our context, advantages of a declarative, modular approach are: ease of specification and maintenance, and opportunities for optimization. AJAX moreover provided an exception handling mechanism, which we plan to consider as well in the future. In the data integration context, systems dealing with data cleaning have been proposed as well. These include [2, 5].

5. CONCLUSION

The advent of XML data, dirty repositories of which already abound, requires adapted cleaning tools. With XClean, we take advantage of the expressive power of XQuery to express XML cleaning programs, while providing to the user (i) a set of modular cleaning operators and (ii) a compact cleaning language (which XClean compiles into XQuery) to specify these operators. The main focus of the demo we propose is on XClean's flexibility, modularity, expressive power, and ease of use, based on a variety of use cases. We also plan to present some interesting XQuery performance issues raised by the kinds of XQuery queries (rich in grouping and function calls) that XClean produces.

Acknowledgements This research was partly funded by a "DAAD Doktorandenstipendium" scholarship.

6. REFERENCES

[1] XClean: A system for declarative XML data cleaning. `http://www.informatik.hu-berlin.de/~mweis/xclean/`.
[2] A. Bilke, J. Bleiholder, C. Böhm, K. Draba, F. Naumann, and M. Weis. Automatic data fusion with HumMer. In *VLDB*, 2005.
[3] S. Chaudhuri, K. Ganjam, V. Ganti, R. Kapoor, V. Narasayya, and T. Vassilakis. Data cleaning in Microsoft SQL server 2005. In *SIGMOD*, 2005.
[4] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, 2005.
[5] A. Fuxman, E. Fazli, and R. J. Miller. ConQuer: Efficient management of inconsistent databases. In *SIGMOD*, 2005.
[6] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, 2001.
[7] J. Hellerstein and J. Naughton. Query Execution Techniques for Caching Expensive Methods. In *SIGMOD*, 1996.
[8] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, May 1995.
[9] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin, Volume 23*, 2000.
[10] V. Raman and J. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, 2001.
[11] D. Shasha, J. Wang, K. Zhang, and F. Shih. Exact and approximate algorithms for unordered tree matching. *IEEE Transactions on Systems, Man, and Cybernetics*, 24, 1994.
[12] P. Singla and P. Domingos. Object identification with attribute-mediated dependences. In *PKDD*, 2005.
[13] M. Weis and F. Naumann. DogmatiX tracks down duplicates in XML. In *SIGMOD*, 2005.
[14] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.
[15] XQuery 1.0. `http://www.w3.org/TR/XQuery`, 2006.