XML Data Integration: Merging, Query Processing and Conflict Resolution

Yan Qi[#1], Huiping Cao[#2], K Selçuk Candan[#3], Maria Luisa Sapino[*1]

[#]*Department of Computer Science and Engineering, Arizona State University, Tempe, Arizona, USA*

[1]*yan.qi@asu.edu,* [2]*hcao11@asu.edu,* [3]*candan@asu.edu*

[*]*Dipartimento di Informatica, University of Torino, Torino, Italy*

[1]*mlsapino@di.unito.it*

ABSTRACT

In XML Data Integration, data/metadata merging and query processing are indispensable. Specifically, merging integrates multiple disparate (heterogeneous and autonomous) input data sources together for further usage, while query processing is one main reason why the data need to be integrated in the first place. Besides, when supported with appropriate user feedback techniques, queries can also provide contexts in which conflicts among the input sources can be interpreted and resolved. The flexibility of XML structure provides opportunities for alleviating some of the difficulties that other less flexible data types face in the presence of uncertainty; yet, this flexibility also introduces new challenges in merging multiple sources and query processing over integrated data. In this chapter, we discuss two alternative ways XML data/schema can be integrated: conflict-eliminating (where the result is cleaned from any conflicts that the different sources might have with each other) and conflict-preserving (where the resulting XML data or XML schema captures the alternative interpretations of the data). We also present techniques for query processing over integrated, possibly imprecise, XML data, and cover strategies that can be used for resolving underlying conflicts.

KEYWORDS

Data Translation, Merging, Conflict-elimination, Conflict-Preserving Integration, Query processing, conflict resolution, query reformulation, compatibility analysis, twig query, feedback

INTRODUCTION

One of the primary motivations behind the development of eXtensible Markup Language (XML) was to create a framework that can support interoperability between businesses and other enterprises. In short time, the simplicity and flexibility of XML led to many new applications, including peer-to-peer (P2P) applications (Koloniari & Pitoura, 2005; Pankowski, 2008), bioinformatics (Achard, Vaysseixm, & Barillot, 2001), and semantic web (Decker et al., 2000).

As we have seen in Chapter titled "XML Data Integration: Schema Extraction and Mapping", the simple, flexible, and self-describing data representation of XML provides unique opportunities to support data integration. On the other hand, these same properties, especially the flexibility of the structure of the data and the possibility for each data contributor and user to have their own schemas also introduce many new challenges in the integration process. Figure 1 provides an overview of the major steps underlying the XML data integration process:

*Figure 1. [Overview of the XML data integration process]. (*Put 'Figure 1.tif' here*)*

- *Schema extraction*: A particular challenge introduced by XML is that not all XML data come with an associated schema. While this enables the use of XML as a flexible messaging and integration medium, when the integration process is schema-aware, it also necessitates a process to extract schemas that can be used during integration.
- *Matching and mapping*: Finding mappings between data components is a common problem in almost all integration domains. XML data can often be represented using trees or tree-like graphs (Goldman & Widom, 1997). This impacts solutions for finding mappings between XML data.
- *XML data/metadata merging*: Once the mappings are discovered, the next step in the process is to integrate the XML data or metadata, depending on whether the system is operating on data- or schema-level. This is often done through a transform-and-merge process.
- *Query processing and conflict resolution*: The results of the merge process, however, may not always be a valid XML data or schema. In these cases, in order to be able to use the resulting merged data in query processing, we either need to apply conflict resolution strategies or develop new query processing techniques that can operate on more relaxed data structures, such as graphs.
  In fact, conflict resolution process can be integrated with query processing to support an incremental approach to cleaning the conflicts: as the user explores the integrated data (and conflicts) within the context of her queries, she can provide more informed conflict resolution feedback to the system.

In Chapter "XML Data Integration: Schema Extraction and Mapping" we have discussed the first two bullets in detail. In this chapter, we focus on merging and query processing over integrated XML data, and cover strategies that can be used for resolving conflicts with the user's help. The running example we use in this chapter is from the same domain (*universities and research institutes*) as Chapter "XML Data Integration: Schema Extraction and Mapping".

MERGING

Once the mappings between the sources are discovered through the matching process, the input sources can be merged into a logical "global" view for further use in integrated data processing. The merge process takes as input (a) a set of sources and (b) the mappings among them, and generates an integrated (target) data or schema.

Handling Conflicts during the Merge Process

Due to the imperfectness of the matching process and possible incompatibilities among the input sources, inputs cannot always be merged perfectly. According to way conflicts are handled, we can classify the merge integration algorithms into two broad categories:

- Conflict-elimination strategies: conflicts are resolved during the integration process and one unified target schema is generated (Pottinger & Bernstein, 2003).
- Conflict-preserving strategies: all interpretations of the data are preserved in one unified representation (Candan, Cao, Qi, & Sapino, 2008; Qi, Candan, & Sapino, 2007a) and conflicts are left to be resolved through user feedback during query processing.

Generalized Mappings

In this section, we will use a more general form of mappings than the one we have used in Chapter titled "XML Data Integration: Schema Extraction and Mapping". Without loss of generality, let us denote each source as a node and edge labeled directed graph, $S(V,E)$, where each node, $v \in V$, corresponds to a labeled element, attribute, or value, and each edge, $e \in E$ between two nodes corresponds to a named relationship between the corresponding data elements. Given this, a mapping over two sources S_1 (V_1,E_1) and S_2 (V_2, E_2) is a pair M $(\mathcal{C}, \mathcal{M})$ where

- $\mathcal{C}$ (C_M, E_M) describes any knowledge that is not directly obtainable from the input sources, such as elements not covered in either source but needed to properly unify the input sources; and
- each element $\mu \in \mathcal{M}$ is in the form of $\langle V_i, V_j, map\_name, \tau \rangle$ where $V_i \subseteq V_1 \cup \mathcal{C}.C_M$, $V_j \subseteq V_2 \cup \mathcal{C}.C_M$, and *map_name* denotes the type of the correspondence (e.g. "*equality*", "*subsumption*", or "*similarity*") between V_i and V_j, and τ is the confidence value associated to this mapping element.

Note that this general mapping definition corresponds to the basic mapping rules defined in the previous section when (a) $\mathcal{C}.C_M$ and $\mathcal{C}.E_M$ are all empty and (b) for $\mu \in \mathcal{M}$, both V_i and V_j contain one element respectively and *map_name* is always "*equality*".

Conflict-Elimination Strategies

In (Pottinger & Bernstein, 2003), Pottinger and Bernstein analyze generic merge requirements for schema integration (or merging), including preserving the elements, relationships, constraints, and properties. More specifically, they present a *Merge* operator for schema integration which satisfies these requirements. The operator works for schemas conforming to a general model, thus can be adapted to XML schemas as well. The algorithm takes as input two source schemas S_1, S_2, and the mapping M_{12} between them. The output is a unified schema S which keeps all the elements and relationships in the input models and the input mapping.

M_{12} is a general mapping as described above, with some limitations:

- First, no confidence value is attached to mapping elements.
- Secondly, V_i and V_j are singleton; i.e., the mapping rules are defined over element pairs.

- The correspondence types are limited to "*equality*" and "*similarity*". "*Equality*" means that two elements are semantically equal, whereas, "*similarity*" denotes that two elements are related but not completely equal.

As discussed in Section "Generalized Mappings", given a correspondence $\mu:\langle v_1,v_2\rangle$, both v_1 and v_2 do not have to belong to S_1 or S_2; but they may be some new concept/element defined in M_{12}. This is, for instance, very useful in representing a mapping correspondence where an element in one schema equals to a combination of several elements in another. As an example, let us consider a scenario where a *name* element in one model refers to two elements *firstName* and *lastName* in another model. This can be represented using three correspondences $\mu_1:\langle c, name\rangle$, $\mu_2:\langle c_1, firstName\rangle$, and $\mu_3:\langle c_2, lastName\rangle$, where c, c_1, and c_2 are new elements introduced, such that c has *parent-child* relationships with c_1 and c_2.

Given S_1, S_2, and M_{12}, the outline of the *Merge* process is as follows:

1. First, the *Merge* operator initializes the integrated schema S with an empty schema.
2. Then, elements are created and added to S. To do this, the elements in S_1, S_2, and M_{12} are grouped, in such a way that there is one group for each mapping condition $\mu:\langle v_i, v_j\rangle$.
3. For each group, a new element is created in S to represent this group of elements. The properties of each new element "c" is the union of the properties of the group that c represents.
4. Next, new relationships are inserted into S. Two cases need to be considered in inserting relationships between two elements c_i and c_j in S, where c_i and c_j represent two distinct groups g_i and g_j, respectively.
    - First case is when g_i and g_j do not contain elements with "*similarity*" type of correspondence. In this case, if there is a correspondence $\mu:\langle v_i', v_j'\rangle$ ($v_i' \in g_i$, $v_j' \in g_j$) of type T and with cardinality l, a new relationship $Rel(c_i,c_j)$ with the same type and same cardinality is created in S for c_i and c_j.
    - If some elements in g_i and g_j have "*similarity*" type correspondence, then a new similarity mapping element, c, is created and every mapping relationship originating from c is replaced by a "*parent-child*" relationship.

Figure 2(a). [(a) Two source schemas and the mapping between them]. (Put 'Figure 2(a).tif' here)
Figure 2(b). [(b)The integrated schema]. (Put 'Figure 2(b).tif' here)
Figure 2. [Example of the Merge *process* (Pottinger & Bernstein, 2003). *Note the edge between two elements denotes the* parent-child *relationship between them. They are implicit in the graphs.].*

Figure 2 shows an example execution of the *Merge* operator. In this example, the element '*edu-university'* in Figure 2(b) merges the nodes '*edu*', '*university*', and 'μ_0' in the source schemas and the mapping. Similarly, '*pname*' in Figure 2(b) incorporates the nodes '*pname*' and 'μ_3' in Figure 2(a). The mapping μ_6 is reflected by creating a new node '*all-bios*' with two children in the original schemas.

Different from (Pottinger & Bernstein, 2003), which eliminates the conflicts and maintains only one merged consistent schema, (Chiticariu, Kolaitis, & Popa, 2008) creates multiple such schemas while one is to be selected in later stages through user interaction. The algorithm first creates a unified representation, where correspondences between elements are represented by mapping-edges. In the second phase, for each compatible subset of mapping-edges, a different merge result is obtained. To cope with the inherent cost of enumerating different mapping strategies, the algorithm uses several heuristics to identify and eliminate redundant strategies.

Conflict-Preserving Strategies

Attempting to resolve conflicts at the merging time may limit the future usage of data: if only one integrated schema is generated by enforcing only some constraints, this integrated target schema obviously misses some of the information in the input sources. In contrast, Candan *et al.* (Candan et al., 2008; Qi, Candan, Sapino, & Kintigh, 2006; Qi, Candan, & Sapino, 2007a) propose to merge sources (schemas or data instances) by preserving the possible different interpretations in the integrated target graph (schema or data instance) and attempt to resolve conflicts only when needed (e.g., in query processing).

A value-null in databases is said to occur when the value cannot be determined for certain. A value-null can be of type "***existential***" (the value exists, but is not known), "***maybe***" (the value may or may not exist), "***place holder***" (the value is known not to exist, so a dummy symbol is used as a place holder), or "***partial***" (the value is known to be in a given set) (Candan, Grant, & Subrahmanian, 1997). For example,

- "*Node &5's tag can be 4, 6, or 9.*"

is a value null.

Qi et al. (2006) introduce structure-nulls, which occur when the structural relationship between the data nodes cannot be determined in certain. For example,

- "*Node &5 is a child of node &3 or &4*".
- "*Either node &5 or &6 is a child of node &3*".

are structure nulls. A structure-null can also be of type "***existential***" (the structural relationship exists, but is not known), "***maybe***" (structural relationship may or may not exist), "***place holder***" (the structural relationship is known not to exist), or "***partial***" (the structural relationship is known to be in a given set of alternatives). An early attempt at modeling semistructured data with missing and partial data is presented in (Liu & Ling, 2000); authors use an object-based model, where *null*, *or-valued*, and *partial set* objects are used to handle partial and missing knowledge in semi-structured data. Although it is richer than standard semistructured data models, such as Object Exchange Model (OEM) (Buneman, Fan, & Weinstein, 1999; McHugh, Abiteboul, Goldman, & Widom, 1997), and Document Object Model (DOM), this model is more focused on *value* nulls and does not capture inconsistencies and missing knowledge in the structure of the data. Qi et al. (2006) present an assertion-based data model, **QUEST**, that captures both *value*-based and *structure*-based "nulls" in data.

Candan *et al.* in (2008) extend this framework to capture more general relationships (e.g., *WORKS-AT*) in addition to XML parent-child relationships. They also extend the assertions with trust values describing how *trustable* each assertion is. The trust value represents the user's source preference, assessment of mapping certainty, and the amount of agreement among different sources on this assertion. Candan et al. (Candan et al., 2008; Qi, Candan, & Sapino, 2007a) also introduce new *coordination* constructs that represent constraints that the integration process imposes on the various source relationships. The six basic constructs introduced in (Candan et al., 2008; Qi, Candan, & Sapino, 2007a), as part of their FICSR framework, are shown in Figure 3. Constructs (a)-(c) coordinate multiple relationships *from* a single element, while constructs (d)-(f) coordinate relationships from multiple elements *to* a single one. Candan et al. (2008) also show how to combine these constructs to obtain more complex and richer coordination semantics among a set of elements. Enriching the integrated graph with these *coordination* constructs (used along with the original source relationships) allows FICSR to preserve the multiple possible worlds (i.e., different interpretations) in the integrated graph. Thus, instead of having to enforce these constraints during the integration time, FICSR is able to defer the resolution process to a later stage in processing. In what follows, we roughly present FICSR's integration process. This work differs in several ways from other integration algorithms, which generally take as input only the source graphs and a mapping. FICSR takes as input, in addition to the source graphs, the following rules:

Figure 3(a). [(a) Negatively coordinate the destinations of w*: either* Rel(w,u) *or* Rel(w,v) *holds, but not both.].* (Put 'Figure 3(a).tif' here)
Figure 3(b). [(b) Positively coordinate the destinations of w*: both* Rel(w,u) *and* Rel(w,v) *needs to hold.].* (Put 'Figure 3(b).tif' here)
Figure 3(c). [(c) Implication of the destinations of w*: if* Rel(w,v) *holds, then* Rel(w,u) *must hold.].* (Put 'Figure 3(c).tif' here)
Figure 3(d). [(d) Negatively coordinate the sources of w*: either* Rel(u,w) *or* Rel(v,w) *holds, but not both.].* (Put 'Figure 3(d).tif' here)
Figure 3(e). [(e) Positively coordinate the sources of w*: both* Rel(u,w) *and* Rel(v,w) *holds together.].* (Put 'Figure 3(e).tif' here)
Figure 3(f). [(f) Implication of the sources of w*: if* Rel(v,w) *holds, then* Rel(u,w) *must hold.].* (Put 'Figure 3(f).tif' here)

Figure 3. [Choice and coordination constructs with (a), (b), (c) destination coordination and (d), (e), (f) source coordination]. (Adapted from (Candan et al., 2008; Qi, Candan, & Sapino, 2007a))

- *mapping rules*: FICSR uses a general mapping rule format; each mapping rule μ: $\langle V_i, V_j, map\_name, \tau\rangle$ consists of subsets, $V_i \subseteq V_1$ and $V_j \subseteq V_2$, of elements from the sources S_1 and S_2, a correspondence name *map_name*, and a confidence τ.
- *embedding rules*: embedding rules of the form ρ:$\langle c_1, c_2$, rel_*name*,$\tau\rangle$ describe how elements that do not belong to any of the sources (but might be created during the integration process) relate to each other. Here, c_1 or c_2 can be either an element created during the integration process or an element in $V_1 \cup V_2$ (but at least one of them is a new element).
- *co-validity rules*: co-validity rules of the form γ:$\langle E, \tau\rangle$ represent the inter-dependence of various relationships in the source graphs; i.e., the relationships in E either all hold together or none holds.

Given mappings and above rules, the merge algorithm integrates the sources by incorporating these mappings and rules:

- For each mapping rule μ:$\langle V_i, V_j, map\_name, \tau\rangle$, a new element c_k with the name *map_name* is created. Then, each element in V_i, V_j is linked to c_k with positively coordinated edges. During this process, the edges to c_k from the elements in V_i (or V_j) are positively coordinated with each other;
- When incorporating an embedding rule ρ: $\langle c_1, c_2, rel\_name, \tau\rangle$ into the integrated graph, if the relationship *rel_name* has no arity constraint, an edge with label *rel_name* is created from the concept c_1 to c_2. Otherwise, the edge may need to be negatively coordinated with other edges. For example, if the edge describes the parent-child

relationship where an element can have only one parent, the edge from c_1 to c_2 with label "*parent*" needs to be coordinated with the other edges coming out of c_1 and going to other concepts.

- To incorporate a co-validity rule of the form $\gamma.\langle E, \tau\rangle$, all the maximal subsets $E' \subseteq E$ with the same source and destinations are detected and all these subsets are positively coordinated. The remaining co-validity requirements are recorded as integrity constraints to be enforced separately.

We use Figure 4 as an example to illustrate the integration algorithm.

Let us consider the following integration rules:

- mapping rules:
  μ_1: $\langle\{lname, fname\}, \{pname\}, president\text{-}name, \infty\rangle$
  μ_2: $\langle\{edu\}, \{university\}, edu(university), \infty\rangle$
  μ_3: $\langle\{name\}, \{name\}, uni\text{-}name, \infty\rangle$
- embedding rules:
  ρ_1: $\langle president\text{-}name, president, parent\text{-}child, \infty\rangle$
  ρ_2: $\langle president\text{-}name, university, parent\text{-}child, 1\rangle$
  ρ_3: $\langle uni\text{-}name, edu(university), parent\text{-}child, 1\rangle$
- no co-validity rules.

While incorporating the three mapping rules, FICSR introduces positive coordinate constructs to connect the source elements to the new concepts.

For example, for μ_1, before coordinating the elements in $\{lname, fname\}$ $\{pname\}$, they are first positively coordinated among themselves. For the embedding rules, FICSR incorporates three corresponding edges in the graph (from *president-name* to *president*, from *president-name* to *university*, from *uni-name* to *edu*(*university*)).

Figure 4(c) shows the full integrated graph.

Note that in the resulting graph, potential conflicts among the input schemas have not been resolved; the identification and resolution of any such conflict is deferred to query processing phase.

Figure 4(a). [(a) First source graph]. (Put 'Figure 4(a).tif' here)
Figure 4(b). [(b) Second source graph]. (Put 'Figure 4(b).tif' here)
Figure 4(c). [(c) Integrated full graph]. (Put 'Figure 4(c).tif' here)
Figure 4. [Example of the FICSR integration process].

While, FICSR data model can capture more general relationships, rich conflicts, and coordination semantics, Kimelfeld & Sagiv (2008) focus on tree structured data. In particular, (Kimelfeld & Sagiv, 2008) introduces probabilistic XML data representation, where XML data are represented in the form of tree structures composed of two types of nodes: ordinary nodes and distributional nodes. Ordinary nodes follow the definition of nodes in the traditional XML model; distributional nodes, however, specify the probabilistic process of generating a random document. Depending upon the semantics indicated by its distribution, a distributional node can be one of the following five types (Kimelfeld & Sagiv, 2008):

- **ind** - the probability of choosing one of its children is independent of that of choosing any other;
- **det** - all of its children are deterministically chosen;

- **mux** - the probabilities of choosing different children are mutually exclusive;
- **exp** - the probability of choosing any of its children is explicitly given;
- **cie** - the probability of choosing a child is determined by the conjunction of a set of independent random Boolean variables, called events.

Kimelfeld et al. (2008) classify probabilistic XML models based on the types of distributional nodes they contain. For instance, $PrXML^{\{ind\}}$ represents probabilistic XML models which use only **ind** distributional nodes. An example of $PrXML^{\{ind,mux\}}$ is shown in Figure 5. There are three distributional nodes: two of them are of type **ind** and the third is of type **mux**. Note that, since they are exclusive with each other, the sum of the probabilities of the two choices under the third distributional node is 1.0.

Figure 5. [An Example of Probabilistic XML Document]. (Put 'Figure 5.tif' here)

QUERY PROCESSING

Query processing over integrated XML data shares many of the key difficulties, which other types of integrated data also pose. For instance, often some form of XML query reformulation is necessary in order to process queries over local data sources. Similarly, when data sources in the XML integration system are autonomous, query processing (even routing (Zhuge, Liu, Feng, & He, 2004)) needs to be performed without the support of a central mechanism, completely through peer-to-peer interaction. Since uncertainty and imperfections can be introduced during source matching and merging, special mechanisms that can handle inconsistency during query processing may be needed. While the special, relatively more flexible structure of XML provides opportunities for alleviating some of the difficulties that other less flexible data types face during integration, it also poses new challenges in that existing XML query processing techniques are often not directly applicable.

As in any integration system, there are two major ways to execute an XML query over data that initially exist in different sources. The first approach is to reformulate the query for each source, execute them independently at these sources, collect results, and integrate these results into a single unified answer. The second approach is to use mappings discovered in the previous steps to integrate the data in a common form and process the query over this integrated data. In this chapter, we discuss both of these two approaches. But, first we provide a brief overview of how queries over XML data are formulated. In Section "XML Query Processing with Local Sources", we discuss major approaches to XML query processing across local sources. Then, in Section "Query Processing over uncertain XML data" we discuss challenges associated to query processing with uncertain data due to imprecise integration.

Querying XML Data

XPath (1999) and XQuery (2006), two popular query languages for querying XML documents, rely on *path expressions* -- which express the desired characteristics of the paths on the underlying data graph -- as building blocks. These path expressions combine requirements about values (such as the element tags of an XML document) with requirements about the structural organization of the elements of interest. Path expressions of type, $P^{\{/,//,*\}}$, are composed of query steps, each consisting of an axis (parent/child "/" or ancestor/descendant "//") test between data elements and a label test (including "*" wildcard which can match any element or tag). Often, multiple path expressions are combined into twigs (i.e., tree patterns (Amer-Yahia, Cho, Lakshmanan, & Srivastava, 2001; Jagadish et al., 2002)) by using path expressions of type $P^{\{[],/,//,*\}}$, where "$[]$" denotes any predicate including sub-path expressions; as illustrated in Figure 6), such tree patterns can be visualized as trees, where nodes correspond to tag-predicates and edges correspond to a parent-child or ancestor-descendant axis. Thus, a twig query, q, can be represented in the form of a node- and edge-labeled tree, $T_q(V_q, E_q)$. The query q may be attached with tag and edge predicates *tag_pred(qv)* and *axis_pred(qe)*, where *tag_pred(qv)* denotes the tag predicate corresponding to the vertex $qv \in V_q$ and *axis_pred(qe)* denotes the axis predicate associated with the edge $qe \in E_q$. An answer to q over data graph G is a pair, $r = \left\langle \mu_{node}, \mu_{edge} \right\rangle$, of mappings:

- μ_{node} is a mapping from the nodes of the query tree to the nodes of the data graph, such that given $qv \in V_q$ and the corresponding data node, $\mu_{node}(qv)$, $tag(\mu_{node}(qv))$ satisfies *tag_pred(qv)*.
- μ_{edge} is a mapping from the edges of the query tree to *simple paths* in the data graph, such that given $qe = \left\langle qv_i, qv_j \right\rangle \in E_q$, the path μ_{qe}, from $\mu_{node}(qv_i)$ to $\mu_{node}(qv_j)$, satisfies *axis_pred(qe)*. Note that a path consisting of a single edge can satisfy both parent-child and ancestor-descendant axis, while a multi-edge path can satisfy only ancestor-descendant axis.

In XPath semantics, for each result instance, $r = \left\langle \mu_{node}, \mu_{edge} \right\rangle$, only the matches for the right most query element are included in the final result (e.g., for the query shown in Figure 6, only the matches for the query element, "grant", are returned).

*Figure 6. [An example query twig]. (*Put 'Figure 6.tif' here*)*

XML Query Processing with Local Sources

In this section, we first discuss major alternatives for XML query processing across local sources. We also present techniques used for XML query reformulation as well as source selection and query routing.

Alternative Architectures for Query Processing with Local Sources

Broadly speaking, there are two alternative architectures for query processing over local sources: the architecture shown in Figure 7(a) makes use of a global (and integrated) XML schema for query reformulation, while the alternative in Figure 7(b) rewrites queries purely, based on peer-to-peer mappings (Cruz, Xiao, & Hsu, 2004; Lenzerini, 2004; McBrien & Poulovassilis, 2003).

- *Query processing with global schema.* The process of query processing with a global schema is illustrated in Figure 7(a). Through a unique interface, or application, an initial query, usually in the form of XPath or XQuery, is constructed on the basis of the global schema (obtained through a priori schema matching and merging). This initial query is reformulated according to the correspondences between the global schema and local schemas. Note that, due to the flexibilities afforded by XML, local schema is often described in XML (through schema extraction in Chapter "XML Data Integration: Schema Extraction and Mapping") even for non-XML data sources. Next, a source selection manager helps identify the data sources over which the reformulated query should be processed. Query optimization and query execution will be performed locally on the selected data sources. Finally, all results are collected from local data sources and combined. Systems exploiting this architecture include MARS (Deutsch & Tannen, 2003a), PEPSINT (Cruz et al., 2004), and Agora (Manolescu, Florescu, Kossmann, Xhumari, & Olteanu, 2000).

- *Query processing through peer-to-peer interactions*. In the peer-to-peer integration approach to XML query processing, mapping rules are created between local schemas using the techniques presented in Chapter "XML Data Integration: Schema Extraction and Mapping", but a global schema does not exist (Figure 7(b)). Again, the local schema is represented in XML. Therefore, the initial query is in the form of XPath or XQuery, described in terms of its corresponding (*target*) schema. A source selection manager helps identify the peers (*sources*) on which the remote query processing should be made. Then, according to mapping rules between target and sources, the initial query is translated into the query formulated in terms of the various source schemas. Systems that follow this approach include Piazza (A. Y. Halevy, Ives, Suciu, & Tatarinov, 2003; A. Y. Halevy et al., 2004), HePToX (Bonifati, Chang, Ho, Lakshmanan, & Pottinger, 2005), and SixP2P (Pankowski, 2008).

Query processing in a peer-to-peer setting can be treated as a special case of processing with global schema, if we think of the target schema as the global schema to which all other local (source) schemas are mapped. This, however, requires $\mathrm{O}(N^2)$ source-to-target mappings, where N is the number of peers, against $\mathrm{O}(N)$ for the case of the former approach. The advantage of the peer-to-peer setting, however, is that (unlike the global schema which may be overly lossy to accommodate all source schemas) the query reformulations may be more precise since it leverages pairwise mappings between peers.

Figure 7(a). [Integration with a Global Schema]. (Put 'Figure 7(a).tif' here)
Figure 7(b). [Peer-to-Peer Integration]. (Put 'Figure 7(b).tif' here)
Figure 7. [Alternative architectures of query processing with local sources].

XML Query Reformulation

In order to process a user's query over local sources, we need to reformulate this query into queries which can be understood by individual data sources: Given a source schema S_S, target schema St, a set of mapping rules, M, between S_S and St, and a query q_t which is defined in terms of St, the goal of query reformulation is to find a query q_s formulated in terms of S_S such that it is equivalent to q_t according to M. Within the context of XML query processing, the initial query q_t, can be in the form of a tree pattern (Arion, Benzaken, & Manolescu, 2007; Calvanese, Giacomo, Lenzerini, & Vardi, 1999; Gao, Wang, & Yang, 2007; Gu, Xu, & Chen, 2008; Lakshmanan, Wang, & Zhao, 2006; Xu & Özsoyoglu, 2005), XPath (Afrati et al., 2009; Balmin, Özcan, Beyer, Cochrane, & Pirahesh, 2004; Cautis, Deutsch, & Onose, 2008; Tang, Yu, Özsu, Choi, & Wong, 2008), or XQuery (Deutsch & Tannen, 2003b; Lenzerini, 2002; Onose, Deutsch, Papakonstantinou, & Curtmola, 2006) statement. Query rewriting schemes differ from each other based on the underlying restrictions on schemas, summaries, and other applicable constraints (such as conjunctive queries only) (Arion et al., 2007; Deutsch & Tannen, 2003b; Lakshmanan et al., 2006). Most generally, we can classify approaches into two major classes based on the way the mapping rules, *M*, are leveraged: global-as-view (GAV, where the global schema is described in terms of the local schemas) and local-as-view (LAV, where each local schema is described as a view over the global schema).

- In GAV, the initial query, stated in terms of the integrated schema S, is translated into queries for local schemas through "view unfolding", where references in the input query to the target schema S are eliminated and replaced by the corresponding references to the local schemas (Gu et al., 2008; Lenzerini, 2002). Consider Figure 8 which presents two local schemas (S_1 and S_2) and a global schema (S_g). The correspondences among these schemas are presented in Table 3(d). Given these and a twig query, "Q_1 = //COLLEGE[NAME]//ELECTRICAL ENGINEERING", the reformulated queries Q_2 and Q_3 are shown in Figure 8(e).
- In LAV, all sources are described as views over the global schema; thus query reformulation can be seen as answering the query using a set of views (A. Y. Halevy, 2001). Most current work on query reformulation for XML integration belongs to this category (Afrati et al., 2009; Arion et al., 2007; Balmin, Özcan et al., 2004; Calvanese et al., 1999; Cautis et al., 2008; Deutsch & Tannen, 2003b; Gao et al., 2007; Lakshmanan et al., 2006; Onose et al., 2006; Tang et al., 2008; Xu & Özsoyoglu, 2005).

Figure 8(a). [Local schema S_1]. (Put 'Figure 8(a).tif' here)
Figure 8(b). [Local schema S_2]. (Put 'Figure 8(b).tif' here)
Figure 8(c). [A global schema S_g]. (Put 'Figure 8(c).tif' here)
Figure 8(d).[Mapping table: each entity in S_g has a view on local schemas].

S_1	S_2	S_g
…	…	…
university	college	COLLEGE
(university.)name	(college.)name	NAME
depart	department	DEPART
CSE	computer science	COMPUTER SCIENCE
EE	electrical engineering	ELECTRICAL ENGINEERING
…	…	…

Figure 8(e). [A twig query in terms of S_g, and its reformulated queries corresponding to S_1 and S_2 respectively]. (Put 'Figure 8(e).tif' here)
Figure 8. [An example of query rewriting in the GAV].

Query reformulation schemes can also be classified into two based on the *qualities* of the resulting reformulations:

- An equivalent rewriting scheme aims to find a rewriting of the initial query *q* with regard to a given view, in a way that preserves the semantics of *q* (Arion et al., 2007; Balmin, Özcan et al., 2004; Cautis et al., 2008; Deutsch & Tannen, 2003b; Onose et al., 2006; Tang et al., 2008; Xu & Özsoyoglu, 2005). Generally, these approaches follow a "generate-and-test" strategy: the initial step produces candidate rewritings, which are then tested to see if they are equivalent to the initial query. A common approach for generating candidate rewritings for conjunctive queries is the "bucket algorithm" (A. Y. Halevy, 2001), which first enumerates possible rewritings for each entity in the query as partial rewritings, then combines these partial rewritings into candidate rewritings for the whole query. Equivalences between the initial query and its rewritings are validated through an unfolding process: (a) first the views in the reformulation are unfolded to express these rewritings in terms of source data and (b) the equivalence between the initial query and these unfoldings are evaluated (Levy, Mendelzon, Sagiv, & Srivastava, 1995).
- Equivalent rewritings may not always exist or may be expensive to identify. An alternative is the *maximally-contained rewriting*, where the reformulated queries do not return all answers, but miss as few of the results as possible. The approaches presented in (Gao et al., 2007; Lakshmanan et al., 2006) fall under this category. (Lakshmanan et al., 2006), for example, first identifies embeddings (i.e., partial matchings from the query to a view in a way preserving node tags and structural relationships) and uses these embeddings to formulate queries on the views. Since the embeddings are potentially more general than the original query, the result set is likely to contain more results than what an equivalent rewriting would return.

In general, even maximally-contained rewritings are not guaranteed to exist. More recently Afrati et al.(2009) proposed *minimally-containing rewritings*, where the results are supersets of those of equivalent reformulations, with only few false additions.

Source Selection and XML Query Routing

Peer-to-peer (P2P) data management systems are gaining in popularity because of their decentralized and distributed nature, which provides a number of advantages, such as high robustness, better use of the resources, better scalability, and the lack of need for integrated-administration (Koloniari & Pitoura, 2005). In peer-to-peer settings, where the search needs to be done in a distributed fashion on multiple peers, being able to quickly locate peers which can help answer a given query is critical for efficiency. This is usually performed in one of the two ways:

- *Source selection*: In this case, each source peer registers its metadata (e.g. schemas), describing its content, into the directory service of the P2P network. In most cases, the registered source description is a summary of the original data or metadata (Cherukuri & Candan, 2008; Tajima & Fukui, 2004). Peers that have queries, then, use these source descriptions to identify peers in the network that have the most relevant schemas or data sets. Source selection approaches include centralized directories (Katsis, Deutsch, & Papakonstantinou, 2008; Mihaila, Raschid, & Tomasic, 2002) as well as distributed directory approaches (Bouchou, Alves, & Musicante, 2003; Cooper, 2004; Nguyen, Yee, & Frieder, 2008).

- *Query routing: A*lternatively, queries are injected into the system and these queries are routed towards peers that have relevant schema/data by the network. The local peers execute queries on their local data and forward the results back to the query originator. Query routing approaches include (Koloniari, Petrakis, & Pitoura, 2003; Koudas, Rabinovich, Srivastava, & Yu, 2004; Peng & Chawathe, 2003; Suciu, 2002; Tatarinov & Halevy, 2004). These often rely on text- or XML-message filtering schemes (Altinel & Franklin, 2000; Candan, Dönderler, Qi, & Kim, 2006; Candan, Hsiung, Chen, & Agrawal, 2006; Diao, Altinel, Franklin, Zhang, & Fischer, 2003; Ives, Halevy, & Weld, 2002) that can quickly route query messages towards relevant peers based on registered source descriptions.

Distributed directory based source selection approaches are generally built on query routing schemes: first, the *source selection query* is routed in the network towards peers that can answer this source selection query. These peers respond back with IDs of peers that are able to answer the main query. Once this phase is over, the initiating peer sends the query to these peers.

Query Processing over uncertain XML data

Uncertainties and conflicts may be introduced during the integration of XML data and metadata. Therefore, processing queries over integrated XML data may require more expressive query processing infrastructures than basic XML frameworks provide. For instance, the mapping rules can be probabilistic in nature and this may lead to integrated XML data which itself is probabilistic. Moreover, conflicts in data sources may render it harder to represent integrated data in tree-like forms which are common to XML; instead, it may be more suitable to leverage graph-based models that are able to describe the inherently more complex structural uncertainty due to integration.

Data Pre-cleaning vs. Pay-as-you-Go

Traditionally, a consistent interpretation (i.e. a "model") of the data or metadata with conflicts is defined as a maximal, self-consistent subset of the data (Bertossi, 2006; Mercer & Risch, 2003). Intuitively, each model is a *possible world*, where there are no conflicts. Data cleaning approaches aim to identify a maximal possible world, which keeps as many of the original assertions about the data (Pottinger & Bernstein, 2003). Restoration of consistency through a model-based interpretation leads to loss of information; thus, identifying a possible world in advance of query processing may be disadvantageous. In such cases, delaying possible-worlds analysis until after query processing might provide context within which conflicts might be eliminated in an informed manner. (Bonifati et al., 2005; Candan et al., 2008; A. Y. Halevy et al., 2003; Qi, Candan, & Sapino, 2007a) and dataspace and pay-as-you-go systems (Dong, Halevy, & Yu, 2007; Franklin, Halevy, & Maier, 2005; A. Halevy, Rajaraman, & Ordille, 2006; Jeffery, Franklin, & Halevy, 2008; Sarma, Dong, & Halevy, 2008) keep alternative plausible interpretations during query processing and assist the user in observing alternatives through a data exploration process at query time.

Data and Result Compatibility

Given an uncertain XML document all results satisfying a query might not be compatible. One way to resolve this problem is to include in the result only those instances that are in all models of the data. This set is often referred to as the set of *certain answers*. (Arenas & Libkin, 2008) shows that computing the set of certain answers for a give query is coNP-complete, except for some special cases. In addition to being expensive to compute, limiting the result to the set of certain answers is often overly cautious. Instead, the *quality* of a result instance can be evaluated based on the amount of conflicts in the data from which it is extracted or based on how compatible it is with the other results to the given query.

Data Compatibility Analysis

As described earlier, (Kimelfeld & Sagiv, 2008) introduced probabilistic XML documents, where each document *P* indicates a set, *D*, of XML documents, called *possible worlds*. Each document d in *D* is associated with a probability, $p(d)$, where *p* is a function to specify the probability distribution of XML documents in *D*. Given a twig query *q*, and a probabilistic XML document *P*, the evaluation of *q* over *P* leads to a set of results, *R*. For each answer $r \in R$, it can be an answer to evaluating *q* over multiple documents in *D*, and its probability (or degree of certainty in terms of possible worlds) is the combination of probabilities of possible worlds related to *q*. Therefore, one way to perform query processing on probabilistic XML document *P* is to enumerate all possible worlds according to *P*, evaluate the twig query *q* over each possible world one by one, and finally compute the probability of each answer. Enumeration however is often intractable, because it is NP-complete to determine if there is a match of *q* in some possible worlds of *P*. Fortunately, the user usually does not need all matches and the top-*K* matches, which have largest probabilities, may be sufficient.

Result Compatibility Analysis

QUEST (Qi, Candan, Sapino, & Kintigh, 2007) captures the compatibility among result instances, a result instance and a set of results or among sets of result, using a reflexive and symmetric "≈" relation:

- Given two result instances r_i and r_j, $r_i \approx r_j$ if and only if the result instances considered together do not violate any structural constraints inherent in XML.
- Given a result instance r' and a set of result instances $R = \{r_1, r_2, \ldots, r_N\}$, $r' \approx R$, if and only if $\forall r_i \in R$, $r' \approx r_i$.
- Given two sets of result instances $R = \{r_1, r_2, \ldots, r_N\}$ and $U = \{u_1, u_2, \ldots, u_M\}$, $R \approx U$ if and only if $\forall r_i \in R$, $\forall u_j \in U$, $r_i \approx u_j$.

Instead of defining the *model* on the data itself, **QUEST** focuses on models of the query results. Given a set of results, R, a compatibility graph, G_c, is used by **QUEST** to capture all pairwise compatibility relationships. (Qi et al., 2006), then defines a *model*, composed of compatible result paths, as a maximal clique in the compatibility graph. For each pair of nodes (representing result paths), an edge is included between them if they are compatible. **QUEST** provides various result exploration options to the user to enable her to obtain a high level understanding of the available data related to her query

The maximal cliques in a graph can be exponential in the number of vertices (Moon & Moser, 1965). There are polynomial time delay algorithms for enumeration of cliques (i.e., if the graph of size *n* contains *C* cliques, the time to output all cliques is bounded by $O(n^k C)$ for some constant *k*) (1988), but in general graphs, *C* can be exponential in *n*; for example as many as $3^{n/3}$ in Moon-Moser's graphs (Moon & Moser, 1965). (Qi et al., 2006) also observes that it is possible to avoid enumeration of cliques or finding of the maximal cliques in the entire compatibility graph, when supporting many of the relevant exploration tasks. For instance, the task of counting the number of maximal cliques a path occurs in can be performed by counting those maximal cliques containing only its neighbors. Also defining the models on the query results as opposed to the data itself, (Qi, Candan, & Sapino, 2007a) is able to significantly reduce the complexity of model-based analysis.

Data and Result Compatibility Analysis

FICSR (Candan et al., 2008; Qi, Candan, & Sapino, 2007a) also performs *model*-based analysis to compute *trust* (or *agreement*) values associated with assertions that make up an integrated data representation. To efficiently compute the agreement values, during an initial off-line analysis process, *FICSR* partitions the integrated relationship graph into small-sized constraint *zones*, each consisting of a mutually-dependent set of relationship constraints. Given a zone, trust value associated with an assertion in this zone is defined in terms of the alternative *models* in which the assertion is valid versus the total number *models* of the zone. Figure 9 illustrates this process with an example. Figure 9(a) is a simplified version of the integrated relationship graph shown in Figure 4(c). In Figure 9(b) this integrated graph is split into six zones; note that the relationship constraints contained in each zone are mutually-dependent. For example, in zone 6, there are two mutually-dependent relationships both of which must exist concurrently in any model. *FICSR* first analyzes each zone individually to obtain an *agreement* score for each relationship alternative. Figure 9(c) shows an example of this zone analysis process: in this example, the agreement value of the relationship between nodes "lname" and "president-name" in zone 6 is computed as 0.5, because this relationship is valid only in one of the two possible models of this zone (see Figure 9(d)). While the agreement analysis process is still NP-complete, the initial zone-partitioning of the graph and the per-zone nature of the agreement analysis prevent this off-line process from becoming unacceptably costly. In *FICSR*, the agreement score of each result is computed based on the agreement scores of the relationships involved in the result; more specifically, given agreement values associated to the underlying assertions, the agreement of a result, r, consisting of assertions, $A(r)$, is computed as

$$agr(r) = \prod_{a_i \in A(r)} agr(a_i).$$

FICSR relies on ranked query processing techniques (Qi, Candan, & Sapino, 2007b) to identify top-*K* high-agreement results to present to the user. While assertions and result agreement values are based on the initial off-line analysis of the integrated data representation, *FICSR* also performs a **QUEST**-like run-time analysis on the results of a given query. In particular, given two results r_1 and r_2 and their assertions $A(r_1)$ and $A(r_2)$, the compatibility between the results are measures in terms of conflicts that assertions in $A(r_1)$ and $A(r_2)$ cause when considered together. If the results identified are found to imply conflicts when considered together, then this leads to the reduction of the *validity* assessments of these results when presented to the user. In particular, when the highest-agreement results are mutually conflicting and thus resulting in low validity, this triggers a feedback process that calls for inputs from the user. The results of the user feedback are reflected on the trust values associated with the assertions in the integrated data as well as the mappings that lead to these trust values to be computed in the first place.

Figure 9(a). [A simplified version of the integrated relationship graph in Figure 4(c)]. (Put 'Figure 9(a).tif' here)
Figure 9(b). [The zone-graph: individual zones in the graph are highlighted with different shades. Note that zones are linked to each other through data/concept nodes.] (Put 'Figure 9(b).tif' here)
Figure 9(c). [After zone analysis, each edge in the integrated relationship graph has a corresponding agreement value.]. (Put 'Figure 9(c).tif' here)
Figure 9(d). [The two models of zone 6]. (Put 'Figure 9(d).tif' here)
Figure 9. [An example of the zone analysis process of FICSR].

Twig Query Processing on Graphs

A structural summary or a merged XML document is often a directed (and weighted) graph instead of being a simple tree. On the other hand, in XML databases, query processors are often designed to exploit the tree-like structure of the XML data. In fact, many existing (binary or holistic) structural join operators, including TwigStack/PathStack (Bruno, Koudas, & Srivastava, 2002), iTwigJoin (Chen, Lu, & Ling, 2005), and Stack-Tree-Desc/Anc (Al-Khalifa et al., 2002), are structurally-informed variants of the standard sort-merge join algorithm: they require that the data nodes are available in a *structurally sorted* order before the join operation can be performed. To implement structural join operations efficiently, most XML query processors rely on index structures based on structurally-informed node labeling schemes (such as Dietz's labeling (Dietz, 1982), which assigns interval-labels to nodes in such a way that descendant nodes have intervals that are contained within the intervals of their ancestors). This enables checking the ancestor-descendant relationships quickly. Such structural labeling and sorting are especially feasible when the underlying data has a tree-structure, but becomes non-trivial when the queries have to be evaluated on graph-data. When data is graph structured, however, these techniques are not directly applicable. (Computer & Vagena, 2004) proposes techniques for evaluating twig queries over graph-structured data. Authors observe that, in a directed graph, the ancestor-descendant relationship of a tree pattern edge is satisfied if there is a path from the ancestor node to the descendant node. Thus, the authors rephrase the ancestor-descendant search in terms of checking *reachability* in the graph and propose a 2-hop cover based labeling scheme (based on (E. Cohen, Halperin, Kaplan, & Zwick, 2002)) to help answer ancestor-descendant queries efficiently (especially on directed acyclic graphs).

Figure 10(a). [A weighted graph fragment]. (Put 'Figure 10(a).tif' here)
Figure 10(b). [One result of the query]. (Put 'Figure 10(b).tif' here)
Figure 10(c). [A second result of the query]. (Put 'Figure 10(c).tif' here)
Figure 10. [A keyword query, {department, grant, professor}, and two matches on a sample weighted graph].

When data have weights, not all results are equally desirable: results need to be ranked according to the underlying cost model. For instance, (Fuhr & Gro\ssjohann, 2001) presents an XML query language extended with IR-related features, including weighting and ranking. XRANK (Guo, Shao, Botev, & Shanmugasundaram, 2003) and ObjectRank (Balmin, Hristidis, &

Papakonstantinou, 2004) compute PageRank (Brin & Page, 1998) style ranking results for keyword-based (IR-style) database queries. XSEarch (S. Cohen, Mamou, Kanza, & Sagiv, 2003), a search engine for XML data, relies on extended information retrieval techniques for ranking. *Retrieval by information unit* (RIU) (W. Li, Candan, Vu, & Agrawal, 2001), BANKS-I (Bhalotia, Hulgeri, Nakhe, & Sudarshan, 2002), BANKS-II (Kacholia et al., 2005), and DPBF (Ding, Yu, Wang, Qin, & Lin, 2007), on the other hand, recognize that in many cases a single node is not sufficient to answer user queries. Instead, given a query consisting of a set of keywords, these algorithms try to find *small* subtrees (in a given weighted graph) containing all the query keywords. An example is shown in Figure 10. In this example, the user provided three query keywords, {department, grant, professor} to be searched on weighted graph fragment in Figure 10(a); here edge weights indicate the cost or penalty of the corresponding edges. In this example, document "a" contains keyword "grant", document "b" contains "department" and document "c" contains "professor". Figure 10(b) shows two results of this query: The result in Figure 10(b) has one more document than that in Figure 10(c), but a smaller total edge cost. Finding minimal trees to answer keyword queries on weighted graphs is shown to be computationally expensive (W. Li et al., 2001). Since users are usually interested in not all but top-*K* results, (Bhalotia et al., 2002; Ding et al., 2007; Kacholia et al., 2005; W. Li et al., 2001) rely on efficient heuristics and approximations for progressively identifying the smallest *K* trees covering the given keywords. As we mentioned above, however, while answering keyword-based queries on graph data is useful in various application domains (such as XML source selection (Aboulnaga & Gebaly, 2007)), for twig query processing, structural relationships between the data elements need to be considered along with keywords and tags (Qi, Candan, & Sapino, 2007b). Thus, using the notation in Section "Querying XML Data", we can define the problem of top-*K* query processing over a given weighted graph G as follows:

- Given a weighted graph G, a query $q = T_q(V_q, E_q)$ and a positive integer K, top-*K* query processing over G is to obtain a set, R, of answers to q over G, in decreasing order of agreement or trust, such that (a) the size of R is K, (b) the i-th answer has higher agreement than the ($i+1$)-th answer, and (c) there are no other answers to q over G having higher agreement than any answer in R.

(Qi, Candan, & Sapino, 2007a) shows that ranked ancestor-descendant relationships (i.e., reachability problem) can be enumerated by applying Yen's top-*K* shortest loopless path algorithm (Yen, 1971). Executing twig queries on the weighted graph, however, requires combining multiple such ancestor-descendant and parent-child results. In the literature, there are a number of ranked-join algorithms for top-*K* queries (Candan, Li, & Priya, 2000; Chaudhuri, Gravano, & Marian, 2004; Fagin, 1996; C. Li, Chang, Ilyas, & Song, 2005). These rely on weight-sorted input streams for pruning unpromising matches. In particular, (Fagin, Lotem, & Naor, 2003; 2003) presents an NRA algorithm which (a) considers data sources which can provide results only in (progressively) descending order of desirability and which (b) enumerates top-*K* desirable join results without having to access all the data from these sources. A common assumption behind all these algorithms, including (Fagin, Lotem et al., 2003), is that the function which evaluates the score of combined results is monotonic. (Qi, Candan, & Sapino, 2007b) develops top-*K* twig query evaluation algorithms for weighted data graphs. In particular, authors present a cost model for the query answers and prove that answering twig queries on weighted graphs is NP-hard. In particular, they show that, while the problem can be viewed as *ranked structural-joins* along query axises, the monotonicity property, necessary for ranked-join algorithms (Candan et al., 2000; Chaudhuri et al., 2004; Fagin, 1996; Fagin, Lotem et al., 2003; Ilyas, Aref, & Elmagarmid, 2003; C. Li, Chang, & Ilyas, 2006), is violated. This is illustrated by the example in Figure 11. The twig query in Figure 6 is first split into sub-queries: "institute//department" and "institute//professor/grant". A match "institute/school/department" to "institute//department" is displayed in Figure 11(a), with cost 12; two matches "institute/school/professor/grant" and "institute/professor/grant" to "institute//professor/grant" are in Figure 11(b), with cost 10 and 9 respectively. The result in Figure 11(c), obtained by combining "institute/school/department" (cost=12) and "institute/school/professor/grant" (cost=10) has smaller overall cost (i.e., 17) than the result shown in Figure 11(d), obtained by combining "institute/school/department" (cost=12) and "institute/professor/grant" (cost=9). The failure of monotonicity in this example is due to the overlapping path fragment "institute/school" between the sub-results that are being combined. Consequently, when processing twig queries, the very common strategy of splitting the twig query into separated path queries, evaluating each path query independently, and then combining sub-results (i.e., results of path queries) with ranked join algorithm cannot be implemented using traditional ranked join algorithms. Instead, authors present a *sum-max monotonicity* property that holds top-*K* twig query evaluation and they develop a new HR-Join algorithm for performing ranked joins efficiently to compute answers to twig queries. (Kimelfeld & Sagiv, 2006) also considers the problem of executing twig-patterns over weighted graphs and proposes polynomial delay (i.e., the time between two consecutive results is polynomial in the size of the input) execution strategies for ranked enumeration of results.

Figure 11(a). [A match for sub-query "institute//department" (cost = 12)]. (Put 'Figure 11(a).tif' here)
Figure 11(b). [Two matches for sub-query "institute//professor/grant" (cost = 10 and 9)]. (Put 'Figure 11(b).tif' here)
Figure 11(c). [A match for query "institute[//department]//professor/grant" (cost = 17)]. (Put 'Figure 11(c).tif' here)
Figure 11(d). [A match for query "institute[//department]//professor/grant" (cost = 21)]. (Put 'Figure 11(d).tif' here)
Figure 11. [An example for ranked structural-join, where the monotonicity property is not satisfied].

FUTURE RESEARCH DIRECTIONS

The existing works in this area have provided promising solutions, but more challenges lay ahead. Pay-as-you-go is a promising strategy towards avoiding the cost of conflict resolution in XML integration. Still, cost of query processing is one of the most significant challenges in XML integration. When the number of involved data sources becomes large or when they are highly conflicting, query processing requires more efficient support at the levels of hardware or software. Parallelizing architectures can be exploited to speed up the query processing through data-partitioned parallel evaluation. New computing frameworks, such as MapReduce (Dean & Ghemawat, 2004) will certainly help in this direction.

Another issue to be considered here can be summarized as the "too-many-answers" problem (Amer-Yahia et al., 2001). When the size of the integrated relationship graph is large, there may be too many results of a given query. It is not possible to display all of them to the user from a practical point of view. Unfortunately, the top-K strategy may not help with this problem, in that not all results are comparable if they do not have scores associated with. Two thoughts can lead to solutions to this problem: Firstly, sampling techniques, where a properly selected sample of the results is presented to the user, can help. Secondly, supported by relevance feedback techniques, query refinement can help user achieve more precise queries to quickly locate information of interest. (Candan et al., 2008; Qi, Candan, & Sapino, 2007a), for example, leverage user feedback for eliminating conflicts identified during query processing. After the system processes the query over data with conflicts and provides a ranked list of results along with highlights showing the conflicts identified within these results, the user is allowed to assess these results and conflicts. The user assessment can be absolute (e.g., "result X is wrong and should be eliminated") or relative (e.g., "I think result X agrees more with my domain knowledge than result Y"). These assessments are used not only to re-rank query results, but also to re-assess (a) importance of constraints that lead to these conflicts, (b) the trust/agreement values associated with the merged data representation (used in computing the query results), and (c) the qualities of mapping rules (which are used for creating the merged data representation in the first place). Feedback driven XML query processing requires further research into (a) the design of easy-to-use interfaces for capturing the user's feedback and (b) algorithms for reflecting the user feedback effectively and efficiently into the conflict resolution and query processing stages of the XML integration workflow.

CONCLUSION

Today, XML is the backbone of all contemporary Web standards and it is increasingly serving as the most ubiquitous data exchange format. While, due to its structural flexibility, in 90's XML gained acceptance as a potential solution to the data interoperability problem, this more flexible nature also implies that there are fewer cues and constraints to inform the integration process. In other words, from one hand, having fewer constraints to deal with implies easier compatibility across data sources; from the other hand, this also implies that there are many more ways to put data together and effective integration requires support from the user. Pay-as-you-go integration, which is becoming more common, is a step in this direction and future research will increasingly focus on techniques that enable context- and user-support to eliminate uncertainties for more effective integration solutions.

References

Aboulnaga, A., & Gebaly, K. E. (2007). μBE: User guided source selection and schema mediation for internet scale data integration. *ICDE,* 186-195.

Achard, F., Vaysseixm, G., & Barillot, E. (2001). XML, bioinformatics and data integration. *Bioinformatics, 17(2)*, 115-125.

Afrati, F., Chirkova, R., Gergatsoulis, M., Kimelfeld, B., Pavlaki, V., & Sagiv, Y. (2009). On rewriting XPath queries using views. *EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology,* Saint Petersburg, Russia. 168-179.

Al-Khalifa, S., Jagadish, H. V., Patel, J. M., Wu, Y., Koudas, N., & Srivastava, D. (2002). Structural joins: A primitive for efficient XML query pattern matching. *ICDE,* 141.

Altinel, M., & Franklin, M. J. (2000). Efficient filtering of XML documents for selective dissemination of information. *VLDB '00: Proceedings of the 26th International Conference on very Large Data Bases,* 53-64.

Amer-Yahia, S., Cho, S., Lakshmanan, L. V. S., & Srivastava, D. (2001). Minimization of tree pattern queries. *SIGMOD Conference,* 497-508.

Arenas, M., & Libkin, L. (2008). XML data exchange: Consistency and query answering. *Journal of the ACM (JACM), 55(2)*

Arion, A., Benzaken, V., & Manolescu, I. a. P.,Yannis. (2007). Structured materialized views for XML queries. *VLDB '07: Proceedings of the 33rd International Conference on very Large Data Bases,* Vienna, Austria. 87-98.

Balmin, A., Hristidis, V., & Papakonstantinou, Y. (2004). ObjectRank: Authority-based keyword search in databases. *VLDB,* 564-575.

Balmin, A., Özcan, F., Beyer, K. S., Cochrane, R. J., & Pirahesh, H. (2004). A framework for using materialized XPath views in XML query processing. *VLDB '04: Proceedings of the Thirtieth International Conference on very Large Data Bases,* Toronto, Canada. 60-71.

Bertossi, L. E. (2006). Consistent query answering in databases. *SIGMOD Record, 35*(2), 68-76.

Bhalotia, G., Hulgeri, A., Nakhe, C., & Sudarshan, S. C. a. S. (2002). Keyword searching and browsing in databases using BANKS. *ICDE,* 431-440.

Bonifati, A., Chang, E. Q., Ho, T., Lakshmanan, L. V. S., & Pottinger, R. (2005). HePToX: Marrying XML and heterogeneity in your P2P databases. *VLDB,* 1267-1270.

Bouchou, B., Alves, M. H. F., & Musicante, M. A. (2003). Tree automata to verify XML key constraints. *WebDB,* 37-42.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks, 30*(1-7), 107-117.

Bruno, N., Koudas, N., & Srivastava, D. (2002). Holistic twig joins: Optimal XML pattern matching. *SIGMOD Conference,* 310-321.

Buneman, P., Fan, W., & Weinstein, S. (1999). Query optimization for semistructured data using path constraints in a deterministic data model. *DBPL,* 208-223.

Calvanese, D., Giacomo, G. D., Lenzerini, M., & Vardi, M. Y. (1999). Answering regular path queries using views. *In Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000,* 389-398.

Candan, K. S., Cao, H., Qi, Y., & Sapino, M. L. (2008). System support for exploration and expert feedback in resolving conflicts during integration of metadata. *VLDB J., 17*(6), 1407-1444.

Candan, K. S., Dönderler, M. E., Qi, Y. a. R.,Jaikannan, & Kim, J. W. (2006). FMware: Middleware for efficient filtering and matching of XML messages with local data.

Middleware '06: Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware, Melbourne, Australia. 301-321.

Candan, K. S., Grant, J., & Subrahmanian, V. S. (1997). A unified treatment of null values using constraints. *Inf.Sci., 98*(1-4), 99-156.

Candan, K. S., Hsiung, W., Chen, S. a. T.,Junichi, & Agrawal, D. (2006). AFilter: Adaptable XML filtering with prefix-caching suffix-clustering. *VLDB '06: Proceedings of the 32nd International Conference on very Large Data Bases,* Seoul, Korea. 559-570.

Candan, K. S., Li, W., & Priya, M. L. (2000). Similarity-based ranking and query processing in multimedia databases. *Data Knowl.Eng., 35*(3), 259-298.

Cautis, B., Deutsch, A., & Onose, N. (2008). XPath rewriting using multiple views: Achieving completeness and efficiency. *WebDB,*

Chaudhuri, S., Gravano, L., & Marian, A. (2004). Optimizing top-k selection queries over multimedia repositories. *IEEE Trans.Knowl.Data Eng., 16*(8), 992-1009.

Chen, T., Lu, J., & Ling, T. W. (2005). On boosting holism in XML twig pattern matching using structural indexing techniques. *SIGMOD Conference,* 455-466.

Cherukuri, V. S., & Candan, K. S. (2008). Propagation-vectors for trees (PVT): Concise yet effective summaries for hierarchical data and trees. *LSDS-IR '08: Proceeding of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval,* Napa Valley, California, USA. 3-10.

Chiticariu, L., Kolaitis, P. G., & Popa, L. (2008). Interactive generation of integrated schemas. *SIGMOD Conference,* 833-846.

Cohen, E., Halperin, E., Kaplan, H., & Zwick, U. (2002). Reachability and distance queries via 2-hop labels. *SODA,* 937-946.

Cohen, S., Mamou, J., Kanza, Y., & Sagiv, Y. (2003). XSEarch: A semantic search engine for XML. *VLDB,* 45-56.

Computer, Z. V., & Vagena, Z. (2004). Twig query processing over graph-structured XML data. *In WEBDB, Paris, Frence,* 43-48.

Cooper, B. F. (2004). Guiding queries to information sources with InfoBeacons. *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware,* Toronto, Canada. 59-78.

Cruz, I. F., Xiao, H., & Hsu, F. (2004). Peer-to-peer semantic integration of XML and RDF data sources. *AP2PC,* 108-119.

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. *OSDI'04: Proceedings of the 6th Conference on Symposium on Opearting Systems Design \& Implementation,* San Francisco, CA. 10-10.

Decker, S., Harmelen, F. V., Broekstra, J., Erdmann, M., Fensel, D., Horrocks, I., et al. (2000). The semantic web - on the respective roles of XML and RDF. *IEEE Internet Computing, 4*, http://www.ontoknow.

Deutsch, A., & Tannen, V. (2003a). MARS: A system for publishing XML from mixed and redundant storage. *VLDB '2003: Proceedings of the 29th International Conference on very Large Data Bases,* Berlin, Germany. 201-212.

Deutsch, A., & Tannen, V. (2003b). Reformulation of XML queries and constraints. *ICDT,* 225-241.

Diao, Y., Altinel, M., Franklin, M. J., Zhang, H., & Fischer, P. (2003). Path sharing and predicate evaluation for high-performance XML filtering. *ACM Trans.Database Syst., 28*(4), 467-516.

Dietz, P. F. (1982). Maintaining order in a linked list. *STOC,* 122-127.

Ding, B., Yu, J. X., Wang, S., Qin, L., & Lin, X. Z. a. X. (2007). Finding top-k min-cost connected trees in databases. *ICDE,* 836-845.

DOM. *Http://www.w3.org/DOM*

Dong, X. L., Halevy, A. Y., & Yu, C. (2007). Data integration with uncertainty. *VLDB,* 687-698.

Fagin, R. (1996). Combining fuzzy information from multiple systems. *PODS,* 216-226.

Fagin, R., Kolaitis, P. G., & Popa, L. (2003). Data exchange: Getting to the core. *PODS,* 90-101.

Fagin, R., Lotem, A., & Naor, M. (2003). Optimal aggregation algorithms for middleware. *J.Comput.Syst.Sci., 66*(4), 614-656.

Franklin, M. J., Halevy, A. Y., & Maier, D. (2005). From databases to dataspaces: A new abstraction for information management. *SIGMOD Record, 34*(4), 27-33.

Fuhr, N., & Gro\ssjohann, K. (2001). XIRQL: A query language for information retrieval in XML documents. *SIGIR,* 172-180.

Gao, J., Wang, T., & Yang, D. (2007). MQTree based query rewriting over multiple XML views. *DEXA,* 562-571.

Goldman, R., & Widom, J. (1997). DataGuides: Enabling query formulation and optimization in semistructured databases. *VLDB}'97,* 436-445.

Gu, J., Xu, B., & Chen, X. (2008). An XML query rewriting mechanism with multiple ontologies integration based on complex semantic mapping. *Inf.Fusion, 9*(4), 512-522.

Guo, L., Shao, F., Botev, C., & Shanmugasundaram, J. (2003). XRANK: Ranked keyword search over XML documents. *SIGMOD Conference,* 16-27.

Halevy, A. Y. (2001). Answering queries using views: A survey. *VLDB J., 10*(4), 270-294.

Halevy, A. Y., Ives, Z. G., Madhavan, J., Mork, P., Suciu, D., & Tatarinov, I. (2004). The piazza peer data management system. *IEEE Trans.Knowl.Data Eng., 16*(7), 787-798.

Halevy, A. Y., Ives, Z. G., Suciu, D., & Tatarinov, I. (2003). Schema mediation in peer data management systems. *In ICDE,* 505-516.

Halevy, A., Rajaraman, A., & Ordille, J. (2006). Data integration: The teenage years. *VLDB},* 9.

Ilyas, I. F., Aref, W. G., & Elmagarmid, A. K. (2003). Supporting top-k join queries in relational databases. *VLDB,* 754-765.

Ives, Z. G., Halevy, A. Y., & Weld, D. S. (2002). An XML query engine for network-bound data. *VLDB J., 11*(4), 380-402.

Jagadish, H. V., Al-Khalifa, S., Chapman, A., Lakshmanan, L. V. S., Nierman, A., Paparizos, S., et al. (2002). TIMBER: A native XML database. *VLDB J., 11*(4), 274-291.

Jeffery, S. R., Franklin, M. J., & Halevy, A. Y. (2008). Pay-as-you-go user feedback for dataspace systems. *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data,* Vancouver, Canada. 847-860.

Johnson, D. S., Papadimitriou, C. H., & Yannakakis, M. (1988). On generating all maximal independent sets. *Inf.Process.Lett., 27*(3), 119-123.

Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., & Karambelkar, H. (2005). Bidirectional expansion for keyword search on graph databases. *VLDB,* 505-516.

Katsis, Y., Deutsch, A., & Papakonstantinou, Y. (2008). Interactive source registration in community-oriented information integration. *Proc.VLDB Endow., 1*(1), 245-259.

Kimelfeld, B., & Sagiv, Y. (2006). Twig patterns: From XML trees to graphs. *WebDB,*

Kimelfeld, B., & Sagiv, Y. (2008). Modeling and querying probabilistic XML data. *SIGMOD Rec., 37*(4), 69-77.

Koloniari, G., Petrakis, Y., & Pitoura, E. (2003). Content-based overlay networks for XML peers based on multi-level bloom filters. *DBISP2P,* 232-247.

Koloniari, G., & Pitoura, E. (2005). Peer-to-peer management of XML data: Issues and research challenges. *SIGMOD Rec., 34*(2), 6-17.

Koudas, N., Rabinovich, M., Srivastava, D., & Yu, T. (2004). Routing XML queries. *ICDE,* 844.

Lakshmanan, L. V. S., Wang, H., & Zhao, Z. (2006). Answering tree pattern queries using views. *VLDB '06: Proceedings of the 32nd International Conference on very Large Data Bases,* Seoul, Korea. 571-582.

Lenzerini, M. (2002). Data integration: A theoretical perspective. *PODS '02: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems,* Madison, Wisconsin. 233-246.

Lenzerini, M. (2004). Principles of P2P data integration. *DIWeb,* 7-21.

Levy, A. Y., Mendelzon, A. O., Sagiv, Y., & Srivastava, D. (1995). Answering queries using views. *PODS,* 95-104.

Li, C., Chang, K. C., & Ilyas, I. F. (2006). Supporting ad-hoc ranking aggregates. *SIGMOD Conference,* 61-72.

Li, C., Chang, K. C., Ilyas, I. F., & Song, S. (2005). RankSQL: Query algebra and optimization for relational top-k queries. *SIGMOD Conference,* 131-142.

Li, W., Candan, K. S., Vu, Q., & Agrawal, D. (2001). Retrieving and organizing web pages by ``information unit''. *WWW,* 230-244.

Liu, M., & Ling, T. W. (2000). A data model for semistructured data with partial and inconsistent information. *EDBT,* 317-331.

Manolescu, I., Florescu, D., Kossmann, D., Xhumari, F., & Olteanu, D. (2000). Agora: Living with XML and relational. *VLDB '00: Proceedings of the 26th International Conference on very Large Data Bases,* 623-626.

McBrien, P., & Poulovassilis, A. (2003). Defining peer-to-peer data integration using both as view rules. *DBISP2P,* 91-107.

McHugh, J., Abiteboul, S., Goldman, R., & Widom, D. Q. a. J. (1997). Lore: A database management system for semistructured data. *SIGMOD Record, 26*(3), 54-66.

Mercer, R. E., & Risch, V. (2003). Properties of maximal cliques of a pair-wise compatibility graph for three nonmonotonic reasoning system. *Answer Set Programming,*

Mihaila, G. A., Raschid, L., & Tomasic, A. (2002). Locating and accessing data repositories with WebSemantics. *The VLDB Journal, 11*(1), 47-57.

Moon, J. W., & Moser, L. (1965). On cliques in graphs. *Israel Journal of Mathematics, 3*(1), 23-28.

Nguyen, L. T., Yee, W. G., & Frieder, O. (2008). Adaptive distributed indexing for structured peer-to-peer networks. *CIKM '08: Proceeding of the 17th ACM Conference on Information and Knowledge Management,* Napa Valley, California, USA. 1241-1250.

Onose, N., Deutsch, A., Papakonstantinou, Y., & Curtmola, E. (2006). Rewriting nested XML queries using nested views. *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data,* Chicago, IL, USA. 443-454.

Pankowski, T. (2008). XML data integration in SixP2P: A theoretical framework. *Intl. Workshop on Data Management in Peer-to-Peer Systems,* 11-18.

Peng, F., & Chawathe, S. S. (2003). Streaming XPath queries in XSQ. *ICDE,* 780-782.

Pottinger, R. A., & Bernstein, P. A. (2003). Merging models based on given correspondences. *VLDB,*

Qi, Y., Candan, K. S., & Sapino, M. L. (2007a). FICSR: Feedback-based inconsistency resolution and query processing on misaligned data sources. *SIGMOD,* 151-162.

Qi, Y., Candan, K. S., & Sapino, M. L. (2007b). Sum-max monotonic ranked joins for evaluating top-K twig queries on weighted data graphs. *VLDB,* 507-518.

Qi, Y., Candan, K. S., Sapino, M. L., & Kintigh, K. W. (2006). QUEST: QUery-driven exploration of semistructured data with ConflicTs and partial knowledge. *CleanDB,*

Qi, Y., Candan, K. S., Sapino, M. L., & Kintigh, K. W. (2007). Integrating and querying taxonomies with quest in the presence of conflicts. *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data,* Beijing, China. 1153-1155.

Sarma, A. D., Dong, X., & Halevy, A. (2008). Bootstrapping pay-as-you-go data integration systems. *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data,* Vancouver, Canada. 861-874.

Suciu, D. (2002). Distributed query evaluation on semistructured data. *ACM Trans.Database Syst., 27*(1), 1-62.

Tajima, K., & Fukui, Y. (2004). Answering XPath queries over networks by sending minimal views. *VLDB,* 48-59.

Tang, N., Yu, J. X., Özsu, M. T., Choi, B., & Wong, K. (2008). Multiple materialized view selection for XPath query rewriting. *ICDE,* 873-882.

Tatarinov, I., & Halevy, A. (2004). Efficient query reformulation in peer data management systems. *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data,* Paris, France. 539-550.

XML. *Extensible markup language: Http://www.w3.org/XML/*

Xpath. (1999). *Http://www.w3.org/TR/xpath*

Xquery. (2006).

Xu, W., & Özsoyoglu, Z. M. (2005). Rewriting XPath queries using materialized views. *VLDB '05: Proceedings of the 31st International Conference on very Large Data Bases,* Trondheim, Norway. 121-132.

Yen, J. Y. (1971). Finding the K shortest loopless paths in a network. *MANAGEMENT SCIENCE, 17*(11), 712-716.

Zhuge, H., Liu, J., Feng, L., & He, C. (2004). Semantic-based query routing and heterogeneous data integration in peer-to-peer semantic link networks. *ICSNW,* 91-107.