XML data integration:Schema Extraction and Mapping

XML Data Integration: Schema Extraction and Mapping

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

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

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

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

[1]mlsapino@di.unito.it

ABSTRACT

Many applications require exchange and integration of data from multiple, heterogeneous sources. eXtensible Markup Language (XML) is a standard developed to satisfy the convenient data exchange needs of these applications. However, XML by itself does not address the data integration requirements. This chapter discusses the challenges and techniques in XML Data Integration. It first presents a four step outline, illustrating the steps involved in the integration of XML data. This chapter, then, focuses on the first two of these steps: schema extraction and data/schema mapping. More specifically, schema extraction presents techniques to extract tree summaries, DTDs, or XML Schemas from XML documents. The discussion on data/schema mapping focuses on techniques for aligning XML data and schemas.

KEYWORDS

Data Integration, Data Extraction, Schema Extraction, Matching, Mapping, Alignment

INTRODUCTION

Data integration is the process of combining multiple heterogeneous and autonomous data sources. Its purpose is to provide a logically unified view of the data to the users who need to search or analyze disparate data sources. Data integration is a well studied problem in the data management community (Doan & Halevy, 2005; A. Halevy, Rajaraman, & Ordille, 2006; Lenzerini, 2002). Despite decades of work in the area, however, the problem is still open. In this chapter, we focus on techniques for eXtensible Markup Language (XML) data integration. As we will see, XML provides opportunities in improving compatibilities across data sources; we will however see that XML also introduces unique challenges that require innovative solutions.

Applications

Many applications require effective and efficient data integration and, as the number and diversity of available data sources increase, this requirement gains further significance. In what follows, we briefly introduce a sample of contemporary applications which require data integration solutions.

- *Data warehousing and business intelligence*. A data warehouse is a repository storing large amounts of data collected from different sources (Devlin & Murphy, 1988). The primary goal of a data warehouse is to provide users unified view of (and efficient access to) data collections that were originally located at different sources. Data warehouses are especially useful in enabling large scale data analysis, for example in support of business intelligence applications. Obviously, unless the contributing data sources are identical in structure or are partitions of a single schema, to build the data warehouse, we first need to integrate the data by identifying the correspondences between the data sources and the data warehouse.
- *Peer-to-peer (P2P) systems*. P2P systems leverage autonomous data sources (peers) as if they are part of a single unified data management system (Koloniari & Pitoura, 2005; Pankowski, 2008). Common usage of such systems includes a user initiating a query through one of the autonomous peer system, but getting answers from all relevant peers. Natural challenges include identifying relevant peers across heterogeneous schema and managing the mappings among the schemas of the peers (Anand & Chawathe, 2004; Cherukuri & Candan, 2008). In addition, queries and answers need to be routed within the peers in the system in a way that eliminates redundant query processing (Anand & Chawathe, 2004).
- *Service oriented architectures (SOA) and web information integration*. Service oriented architectures abstract recurring (e.g., business) activity flows, make them available as independent services, and leverage these services as modules within large software systems. This approach reduces costs of developing and deploying new applications and promotes reuse. Consequently, today, the "web" is not only a collection of hyperlinked pages, but rather a collection of dynamic services that one can use to develop web-based applications and mashups (Jhingran, 2006). These web services, with their descriptions, are published so that other people can locate and integrate them into end-to-end information products. Meanwhile, data spaces (Franklin, Halevy, & Maier, 2005; A. Y. Halevy, Franklin, & Maier, 2006) help reduce the cost of managing loosely structured Web data by eliminating the need to impose strict structures on the integrated data. These, however, requires resolving potential differences between the data service interfaces and underlying data structures.
- *Scientific data management*. In many scientific domains (e.g., archeology (Kintigh, 2006) and biology (Achard, Vaysseixm, & Barillot, 2001)), individual researchers or communities have different data management conventions, standards, and taxonomies (Qi, Candan, & Sapino, 2007). For example, bioinformatics data have many new data types (e.g., microarrays, interaction maps of proteins, etc.) stored in different databases and in different formats (Achard et al., 2001) . In archaeology, there is almost no universally agreed structure or ontology to help support integration and eliminate conflicts that occur due to varying knowledge standards and data interpretations (Kintigh, 2006).

Why XML?

In mid 90's, the growing need for a common platform that can provide uniformity and improve interoperability between businesses and other enterprises led to the wide acceptance of eXtensible Markup Language (XML) as an exchange framework. Today, most of the data interchange is

through XML-based data representation standards. XML provides simple, flexible, and self-describing data representation. Its flexibility is due to the fact that alternative schemas can be combined effectively using *disjunctions*. Moreover, it is self-describing in that XML instances carry the structure of the data in the form of human-readable tags that are associated with data elements; consequently XML data can be exchanged without associated schemas. This simplicity and flexibility led to XML's use in many different domains for which ease of data exchange is a primary requirement, these include peer-to-peer (P2P) applications (Pankowski, 2008), bioinformatics (Achard et al., 2001), and semantic web (Decker et al., 2000).

On the other hand, these same properties, especially the flexibility of the structure of the data and the possibility for each user or data contributor to have their own schemas (through Document Type Definitions (DTDs) and XML schemas (E. Rahm, Do, & Massmann, 2004) as opposed to committing to a unique, fixed set of constraints constraining the organization of the data, introduce new challenges in the integration process (Bertino & Ferrari, 2001) . In late 90's, Halevy (1999) investigated issues that were then considered critical for XML data integration, including the choice of suitable languages for the description of data sources, the definition of query reformulation algorithms, the translation among different Document Type Definitions (DTD's), and the formulation of formalisms for source descriptions.

During the past decade, two key complementary challenges to XML data integration emerged: (a) finding alignments, similarities, and compatibilities between different XML data schemas or instances and (b) identifying and resolving conflicts between XML data sources whenever they are not compatible.

Figure 1
[Put figure fig1.tif here]
[Figure caption: Overview of the XML data integration process]

Outline of the Chapter

This chapter will focus on the challenges and solutions in the XML data integration. Figure 1 provides an overview of the underlying process:

- *Schema extraction*: A particular challenge introduced by XML is that not all XML data come with an associated schema. In fact, one of the major differences between XML and its predecessor Standard Generalized Markup Language (SGML) is the relaxation of the requirement of each document having an associated document type definition (DTD), which defines the rules governing the structure. While this enables the use of XML as a flexible messaging and integration medium, in some cases (especially when the integration process is schema-aware), it also necessitates a process to extract a schema from a given collection of schema-less XML documents. We discuss this in Section “SCHEMA EXTRACTION”.
- *Matching and mapping*: Finding mappings between data components is a common problem in almost all integration domains. For example, multi-tenant databases, which form the core of many Software and Information as a Service solutions (Aulbach, Grust, Jacobs, & Rittinger, 2008), strive to create integrated/consolidated schemas across similar, but different, tenant schemas. The mappings from tenant schemas to the consolidated schema help the system manage multiple tenants as a single tenant, thus reducing the overall management and maintenance cost. XML data can often be

represented using trees or tree-like graphs (Do & Rahm, 2002; Goldman & Widom, 1997). This impacts solutions for finding mappings between XML data. We discuss XML matching and mapping methodologies in Section "MATCHING AND MAPPING".

- *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.
- *Query processing and conflict resolution*: The results of the merge process, however, may not always be a valid XML data or schema. This step uses the resulting merged data to support query processing and apply conflict resolution strategies.

In this chapter, we focus on the first two steps. In a separate chapter, titled "XML Data Integration: Merging, Query Processing and Conflict Resolution", we will discuss the later two steps, merging, query processing over integrated XML data, and the strategies that can be used for resolving conflicts. Finally, we conclude the chapter in Section "CONCLUSION".

Running Example

All the examples presented to illustrate the algorithms in this chapter are picked from *universities and research institutes* application domain, where the underlying data includes

- funding organization information, e.g., organization name, organization location, and title of grants (or funds);
- university information, e.g., university name, and information about the university president; and
- faculty information, e.g., faculty name and his/her funding information.

SCHEMA EXTRACTION

While in many cases XML documents are created according to a pre-defined structure (e.g., Document Type Definition (DTD) or XML schema (XMLschema)), the existence of a DTD or a schema is not guaranteed. In fact, it has been observed that many XML documents on the web do not follow explicit schemas (Barbosa, Mignet, & Veltri, 2005) (i.e., schemas are unavailable (Barbosa, Mignet, & Veltri, 2006), or the existing schemas are not valid (Bex, Martens, Neven, & Schwentick, 2005). However, during XML data integration (especially when integration needs to be supported by mappings extracted from schemas (E. Rahm & Bernstein, 2001; Shvaiko & Euzenat, 2005), it is critical to have schema information in advance. These lead to research on learning the (implicit) structure of a given XML corpus through various structure extraction techniques (Florescu, 2005).

Intuitively, on one hand, the extracted schema must represent all of the input XML documents (this is referred to as the *generalization* property); i.e., each input document must be an instance of the extracted common schema. On the other hand, we do not want to extract an *overly-general* schema, which covers significantly more XML documents than the input data; in other words, the extracted schemas should be specific enough to cover only the input XML documents. This is referred to as the *specification* property. With these properties in mind, we can define the schema extraction problem as follows (Bex, Neven, Schwentick, & Tuyls, 2006; Bex, Neven, & Vansummeren, 2007; Garofalakis, Gionis, Rastogi, Seshadri, & Shim, 2003; Goldman & Widom, 1997):

Definition. The ***schema extraction*** problem is to identify a schema *S* from a given set of XML documents *D* such that *S* captures the structural information of the documents in *D* in a minimal way. (I.e., *S* is general and specific enough at the same time to cover *D*).

The schema extraction process is also referred to as *schema inference*. The underlying structure of a given collection of XML documents can be described using DTD, XML Schema, or in a more general representation such as tree or graph. The structure extraction techniques in the literature target at inferring three kinds of representations: tree or graph summaries (Goldman & Widom, 1997), DTDs (Bex et al., 2005; Bex et al., 2006; Garofalakis et al., 2003), or XML Schemas (Bex et al., 2007; Bex, Neven, & Vansummeren, 2008; Hegewald, Naumann, & Weis, 2006). This section considers these different approaches and discusses representative techniques.

Extraction of Tree and Graph Structures

If one ignores the explicit object references, an XML data/document has a hierarchical structure. OEM (Papakonstantinou, Garcia-Molina, & Widom, 1995) and LORE (McHugh, Abiteboul, Goldman, & Widom, 1997) are two well-known tree-like data models for XML documents that leverage the hierarchical nature of XML data. In OEM, for example, database is a rooted, directed graph, with textual labels on edges and atomic values in leaves. More specifically, each node of the graph corresponds to an element or an attribute of an element in the XML document. A child node corresponds to a sub-element or an attribute of its parent node. For each child of a node, besides the pointer to the child, there is a tag that indicates the name of the child node. If the child is a sub-element, the name is its element tag. If the child is an attribute, the name is the attribute name. In fact, a collection of XML documents can also be viewed as a single large DOM tree, where all the individual documents are rooted at the same node. Figure 2(a) provides an example. Given such an OEM tree, the schema extraction problem can be posed as understanding the common structure governing the root to leaf paths on this tree. Based on this observation, in (Goldman & Widom, 1997), Goldman and Widom present the *DataGuide* technique for extracting structures as concise summaries of initially schema-free XML databases.

A *DataGuide* of an XML database is a graph, where each object node has a unique identifier and nodes are linked with directed labeled edges. Unlike the graph corresponding to an XML document, each label path (between a given pair of nodes) in the DataGuide is unique. A DataGuide is said to represent a given XML database, if there is one and only one label path for each path in the XML database. More formally, let *DG* be a DataGuide extracted from a database *D* represented in OEM:

- every label path in the database D exists in the DataGuide *DG* (covers all structural information),
- every label path in the DataGuide *DG* exists in *D* (no redundant information), and
- let *tg*(*l*) denote the set of target objects one can reach starting from the root and following a label path *l*. In an XML database *tg*(*l*) may contain more than one object, but the target set of a label path in a DataGuide is a singleton set (minimality).

Figure 2
[Put figure fig2a.tif here]
[Insert Figure 2a's subtitle: (a)]
[Put figure fig2b.tif here]
[Insert Figure 2b's subtitle: (b)]

[Put figurefig2c.tif here]

[Insert Figure 2c's subtitle: (c)]

[Insert Figure 2's caption: Example for DataGuide: (a) a database, (b) a tree structured data guide, and (c) a graph structured dataguide]

Figure 2(b) shows a DataGuide for the sample database in Figure 2(a). In the original database, between object nodes '*1*' and '*9*', there is a label path *org.name*; there is a similarly labeled path (between object nodes '*14'* and '*17*' in the DataGuide and this label path is unique.

Note that a given XML data collection may have more than one DataGuide. The graphs in Figures 2(b) and 2(c) are both DataGuides of the database in Figure 2(a). A strong DataGuide (Goldman & Widom, 1997) is a DataGuide such that two label paths l_1 and l_2 point to the same object (i.e., their target sets are identical) if and only if the target sets of l_1 and l_2 are exactly the same in the original database. More formally, given label path l, let $L_{DG}(l)$ be the set of label paths in the DataGuide *DG* which have the same target set as l. Similarly, let $L_D(l)$ be the set of label paths in the Database *D* which have the same target set as l. A *strong DataGuide* refers to a DataGuide such that $\forall_l L_{DG}(l) = L_D(l)$. Consider Figure 2 as an example and let l be *org.name*. The target set of l in *D* is {5, 9} and the set of label paths sharing this target set is $L_D(l)$ = {*org.name*}. Consider the same label path in the DataGuide in Figure 2(b); here the target set of l is {17}, the only label path that reaches this target set is *org.name*; thus, we have $L_{DG}(l)$={*org.name*}=$L_D(l)$ = {*org.name*}. After similar analysis on other label paths, we can see that Figure 2(b) is a strong DataGuide of Figure 2(a). Figure 2(c), however, does not show a strong DataGuide. To see this, consider the same label path *org.name*; in this case, *tg*(l) in the DataGuide is {29} and $L_{DG}(l)$={*org.name*, *edu.name*}, which is different from $L_D(l)$. Thus, the graph is not a strong data guide.

While DataGuides are not unique, Goldman & Widom (1997) show that each XML collection has one and only one strong DataGuide; moreover, given a tree-structured XML collection, construction of the corresponding strong DataGuide is linear in space and time, with respect to the size of the collection. For a more general database (with explicit references, creating a graph), however, the process has exponential cost (intuitively, DataGuide construction is equivalent to conversion of a non-deterministic finite automaton into deterministic one). Strong DataGuides are created by performing a depth-first traversal of the database and recording the target sets of the label paths visited. Since in strong DataGuides there is a one-to-one correspondence between *source target sets* (the target sets of label paths in the source) and DataGuide objects, the algorithm presented in (Goldman & Widom, 1997) maintains a hash table *H* to keep the one-to-one correspondences between the target sets in the source database and the objects in the DataGuide for examined label paths. To begin with, a DataGuide object is created to correspond to the root of the source database. That is, an entry ({$root_D$}: $root_{DG}$) is inserted into *H*. Then, the unexamined source target sets in *H* are expanded in a depth-first order to compute new source target sets and to create corresponding DataGuide objects for them. To expand a source target set *tg* (for an entry (*tg* : *o*) in *H*), the algorithm first gets the labels coming out of any object in *tg*. Then, through different labels, it gets their reachable source target sets. Once a newly computed source target object set *tg'* does not exist in *H*, a new DataGuide object *o'* is created for *tg'* (i.e., a new entry, (*tg'* : *o'*), is added to *H*) and a link from object *o* to *o'* with label l is created. Otherwise (i.e., an entry (*tg'* : *o'*) exists in *H*), the algorithm simply links *o* to *o'* using label l. Take the database in Figure 2(a) as an example. Initially, *H*={({*1*}:*14*)}. Next, the target set {*1*} is expanded by following two different labels *org* and *edu*. Following the first label *org*, we get

source target set {*2*, *3*}. Since this does not exist in *H*, we need to create a corresponding DataGuide object, "*15*", for it and insert this information to *H*, then *H* has one more entry ({*2*, *3*}:*15*). This process is recursively applied to source target set {*2*, *3*}, and so on. When every target set is expanded for label paths starting with *org*, the expansion continues to label paths starting with *edu*. Finally, this algorithm would result in the strong DataGuide shown in Figure 2(b).

Since, especially for graph structured databases, strong DataGuide construction can be exponential, Jennifer & Widom (1999) introduce *approximate data guides* that can reduce the construction cost by relying on approximate hash matches. T-Index (Milo & Suciu, 1999) is also similar to DataGuides, but the paths represented in a T-index structure are not limited to those starting from the root. APEX (Chung, Min, & Shim, 2002) is similar to DataGuides and T-Indexes, but it extracts the structure only for frequent paths in the data.

DTD and XML Schema Extraction and Inference

As described above, DataGuides use graphs as the general form to represent the common structures of XML collections. DataGuides, however, are not as expressive as Data Type Definitions (DTDs) (Bex et al., 2005; Bex et al., 2006; Chidlovskii, 2001; Garofalakis et al., 2003; Min, Ahn, & Chung, 2003; Sankey & Wong, 2001) or XML Schema (Bex et al., 2007; Bex et al., 2008; Clark; Hegewald et al., 2006), the two most widely used formats to represent the structure of XML documents.

Figure 3

[Put figure fig3a.tif here]

[Figure 3a subtitle: (a) Initial SOA created for the XML document in Figure 2(a)]

[Put figure fig3b.tif here]

[Figure 3b subtitle: (b) Step 1: apply the *optional* rule on (a). The edge from "*location*" to "s_{end}" is removed; The state "*state*" is changed to "*state?*"]

[Put figure fig3c.tif here]

[Figure 3c subtitle: (c) Step 2: apply the *concatenation* rule on states labeled with "*location*" and "*state?*" on (b)]

[Put figure fig3d.tif here]

[Figure 3d subtitle: (d) Step 3: apply *disjunction* rule on (c) by merging the states "*fund*", "*grant*", and "*location state?*"]

[Figure 3 caption: Transformation of a single occurrence automaton in (a) towards a single occurrence RE]

DTD Extraction

DTD's are most often used for grammar validation, which is the process through which a service verifies the validity of a document against a registered DTD to ensure that it is structurally valid and processable. A DTD can be formally abstracted as quadruple Γ (I, T, Rt, Φ), where I is the set of non-terminals, $T = \Sigma \cup \bar{\Sigma}$ is the set of terminals (Σ denotes the alphabet of open-tags and $\bar{\Sigma}$ denotes the set of close-tags), Rt is the root, and Φ is a set of production rules which can be used to generate XML documents matching the given DTD. In particular, the strings represented by the right-hand sides of the production rules in Φ are regular expressions and, thus, can be recognized by finite state machines (Hopcroft & Ullman, 1979). Balmin, Papakonstantinou, &

Vianu (2004), for example, use this fact to develop an incremental, divide-and-conquer type of validation mechanism for XML documents. Many works (Chitic & Rosu, 2004; Gottlob, Koch, Pichler, & Segoufin, 2005; Segoufin & Vianu, 2002) also show that XML documents can be validated using finite state automata. Based on the observation that DTDs can be abstracted as regular expressions (REs) that can be recognized by finite state automata, a number of works focus on learning these REs. Bex et al. (2006) for example, reduce the DTD extraction problem to learning REs from XML fragments in the XML corpus. Garcia & Vidal (1990) also derive an RE to represent a given XML corpus. Several researchers observed that REs learned through these processes tend to be overly complex to be useful in practical settings (Ehrenfeucht & Zeiger, 1976; Fernau, 2004; Fernau, 2005). Bex, Neven, & Bussche (2004) observe that a significant majority (99% of XSDs or DTDs in practical use) can be represented as single occurrence REs (SOREs), where every element name occurs at most once in the expression. For example, "$((b?(a|c))^{+}d))^{+}e$" is a SORE while "$a(a|b)^{*}$" is not ("a" occurs more than once).

Relying on this observation, Bex et al. (2006) provide a scheme to generate SOREs from XML data. A SORE, however, may not be found if the DTD corresponding to the given XML collection cannot be represented using an expressions where each element name occurs only once. In such cases, Bex et al. apply heuristics to find SOREs that are less accurate (i.e., corresponding to a more general DTD than the given XML collection implies). The process is as follows:

- In the first step, the algorithm collects all label paths from root to leaves. In (Bex et al., 2006), these label paths are called strings. For example, from the XML document in Figure 2(a), we can extract strings *org.name* and *org.location.state*.
- The second step constructs a so called, *single occurrence automaton* for these strings. The automaton has two special states s_{init} as the starting state and s_{end} as the terminal state. All the other states are labeled with element names. There is an edge from one state s_j to another state s_k if (i) $s_j = s_{init}$ and s_k is a starting element name in some XML fragment string, or (ii) $s_k = s_{end}$ and s_j is an ending element name in some XML fragment string, or (iii) $s_j\ s_k$ is a 2-gram extracted from some XML fragment string. Figure 3(a) shows the automaton we can obtain using the XML fragments in Figure 2(a).
- In the third step, this initial single occurrence automaton is simplified to obtain the SORE (if it exists) by applying four transformation rules:
    - *disjunction rule* merges the states which share the same predecessors and successors disjunctively (using "|"),
    - *concatenation rule* concatenates those adjacent states having only one incoming and outgoing edge,
    - *self-loop rule* removes any self-loop edge on a state "s" and relabels it as "s^{+}",
    - *optional rule* removes such edges $s_i \rightarrow s_j$ that there exists another state s_k with $s_i \in prec(s_k)$ and $s_j \in succ(s_k)$. In this case, s_k is relabeled as $s_k^{?}$.

Figure 3 gives an example illustrating how these rules are used. When we reach Figure 3(d), the algorithm stops since none of the above rules can be applied. However, Figure 3(d) is not an RE yet.

- If a SORE cannot be derived using the above rules, the algorithm falls back onto repairs rules that allow some fuzziness in merging the automaton states. For example,
    - an *enable-disjunction rule* is used to add a minimal number of edges to make the predecessors and successors of two states are the same.

In Figure 4, this rule adds the bold line from state "*edu*" to state "*location state*$^{?}$|*fund*|*grant*". Given this new edge, *disjunction* and *concatenation* rules can now be applied to obtain the regular expression in Figure 4(b) and 4(c).

Figure 4
[Put figure fig4a.tif here]
[Figure 4a subtitle: (a) Apply *enable-disjunction* rule on Figure 3(d)]
[Put figure fig4b.tif here]
[Figure 4b subtitle: (b) Apply *disjunction* rule on (a)]
[Put figure fig4c.tif here]
[Figure 4c subtitle: (c) Apply *Concatenation* rule on (b)
[Figure 4 caption: SOA repair]

XML Schema Extraction

XML Schema Definitions (XSDs) are more expressive than DTDs. In particular, XSD introduces types, which are essentially regular expressions (Martens, Neven, Schwentick, & Bex, 2006) that can be used to describe elements in the XML schema; each element can take one or more types. The fundamental difference from DTDs, however is that the type (or the corresponding RE) an element will take may be determined by the context (i.e., ancestor elements) in which the elements occur in the XML document. This is in stark contrast to DTDs, where given a disjunctive element definition of the form $A := RE_1|RE_2$, there is absolutely no constraint on whether A can be expressed in a given document using RE_1 or RE_2. In XSD, however, the choice between RE_1 and RE_2 can be tied to the ancestors of A in the given document.

Since DTD extraction techniques do not look for such dependencies, given a set of documents that are created using XSDs (where such dependencies exist), DTD extraction techniques will fail to find them. Bex et al. (2007) argue that in many of the existing XSD extraction approaches, such as Trang (Clark) and XStrut (Hegewald et al., 2006), while the extracted schemas are in XSD syntax, they are equivalent to DTDs in expressive power.

When inferring a DTD, since there is no contextual dependence, the algorithm only needs to distinguish the immediate parent-child relationship among the XML tags to learn the REs corresponding to elements. In learning XSDs, on the other hand, the algorithm also needs to seek to identify whether contexts (i.e., the root-to-element paths in the given set of document) have any impact on the REs corresponding to each element name. This increases the complexity of the analysis. Due to this inherent complexity, Bex et al. (2007) focus on learning a subclass of XSDs commonly used in practice. XSDs in this subclass, named *k-local single occurrence XSD*s (SOXSDs), satisfy the following two properties: First, in these XSDs, determining one element's content model (e.g., "*name*" defined under "*org*") only depends on a limited number (e.g., *k*) of ancestors of the element -- this is based on the study in (Martens et al., 2006) that, in 98% of the XSDs, one element's content model can be determined based on the label of itself, its parent, or its grandparent (i.e., up to *k=2*); Second, these XSDs only contain elements with different names as in *SORE*s discussed above.

MATCHING AND MAPPING

Matching is a vital step in XML data integration. Given schemas of separate data sources, the matching operation discovers the correspondences or mappings (among constituent objects, such as attributes or values, from different sources) that are not immediately available (for example,

due to differences in naming convention) (E. Rahm & Bernstein, 2001). A multitude of approaches (Do & Rahm, 2002; Fuxman et al., 2006; Gal, 2007; Hernández et al., 2007; Hernández, Papotti, & Tan, 2008; Madhavan, Bernstein, & Rahm, 2001) have been developed to perform matching operation for different types of data and metadata. . In the context of XML data integration, the term *matching* applies to finding correspondences among XML schemas (including DTDs) or document instances (XML documents). The techniques focusing on relational database schema matching or ontology matching will not be our focus, but we will refer to them when they are closely related to XML matching. After obtaining the matching results, further processing is needed to translate these correspondences to executable scripts (e.g., SQL, XQuery). Some existing works (Atay, Chebotko, Lu, & Fotouhi, 2007; Fuxman et al., 2006; Hernández et al., 2007; Hernández et al., 2008; Pankowski, Cybulka, & Meissner, 2007; Popa, Velegrakis, Miller, Hernández, & Fagin, 2002) call these scripts *mappings* and call this process *mapping generation*. In order to distinguish the results of the basic matching operation and this one, we call the results of this operation *mapping rules* and call this operation *mapping rule generation*. These mapping rules are used for performing further operations, e.g., data exchange (Hernández et al., 2008), data translation (Milo & Zohar, 1998; Popa et al., 2002), or query evaluation (chapter titled "XML Data Integration: Merging, Query Processing and Conflict Resolution").

In what follows, we first define the terminology in Section "Terminology". Then, in Section "Matching Operation: Identifying Mappings", we detail some main challenges in schema matching and some typical techniques. We cover the problem of mapping rule generation in Section "Mapping Rule Generation"

Terminology

Based on the characteristics of the underlying data and metadata, various types of matching techniques have been developed: these include schema matching (e.g., relational database schemas, catalogs, and XML schemas), ontology matching (Shvaiko & Euzenat, 2005; Shvaiko & Euzenat, 2008), and so forth. Unfortunately, the terminologies used for denoting this operation, such as *matching* (Cupid (Madhavan et al., 2001), COMA(Do & Rahm, 2002)), *match* (Cupid), *alignment* (QOM (Ehrig & Staab, 2004)), *mapping* (QOM (Ehrig & Staab, 2004)), differ from context to context. For clarity, in this chapter, we use the term *matching* to denote the operation, and use *correspondences*, *mapping* or *alignment* to denote the results of this operation.

As stated above, given two data sources S_1 and S_2, the matching operation identifies correspondences between parts (e.g., elements or element sets) of S_1 and S_2. Each such correspondence has an associated confidence value (or probability) $\tau(\in[0,1])$ (the correspondence is more certain when this value is closer to 1). Many works (e.g. (Madhavan et al., 2001)) use the convention that a mapping denotes the matching result as a whole, while a correspondence refers to one pair of matched elements. We also follow this convention. Thus, the set, $M = \{\mu\}$, of correspondences is called a *mapping* (some work (Fuxman et al., 2006) also call the matching results as *matchings*). The *mapping rules*, on the other hand, are pieces of scripts written in specific languages (e.g., SQL, XQuery, etc.) to reflect a mapping between two sources. The mapping rule is sometimes referred to an assertion (Candan, Cao, Qi, & Sapino, 2008; Qi et al., 2007). Each mapping rule from S_1 to S_2 specifies how relevant parts of S_1 can be translated to a form compatible with S_2. Potential uncertainties in mappings (i.e., cases where $\tau < 1.0$) leads to the following observations:

- *Non-singleton mapping sets*. A part of S_1 may match multiple parts of S_2, with different confidence values. Depending on the semantics of the parts being mapped and the integrity constraints governing S_1 and S_2, these mappings might be compatible with each other or conflicting. When only one mapping is allowed, often only the most likely correspondence is maintained (COMA (Do & Rahm, 2002), LSD(Doan, Domingos, & Halevy, 2001), Cupid (Madhavan et al., 2001)). Otherwise, by picking correspondences whose confidence values exceed some threshold, one-to-many mappings can be preserved (Cupid).
- Matching results are not always symmetric. That is, the results of matching S_1 to S_2 may be different from the results obtained from matching S_2 to S_1 (Do & Rahm, 2002)
- Matching similarity is not necessarily transitive. Let v_1, v_2 and v_3 be three elements in sources S_1, S_2, and S_3. Let also $v_i \rightarrow v_j$ denote a correspondence identified from v_i to v_j If $v_1 \rightarrow v_2$ and $v_2 \rightarrow v_3$, then in general there is no guarantee that $v_1 \rightarrow v_3$.

This of course is a potential problem as it may result in semantically inconsistent scenarios. While it is hard to avoid this problem with mappings identified through pairwise matching operations, composite and hybrid matching techniques may avoid it by considering more than two pairs at a time (COMA (Do & Rahm, 2002)).

For simplicity of the discussion, in the rest of this section, we will focus on *mapping*s of the form $\{\mu: v_i \rightarrow v_j\ (\tau_{ij})\}$, where v_i and v_j are two elements from two different sources S_1 and S_2. However, since during data integration, algorithms may need to take as input more general mappings (Candan et al., 2008; Pottinger & Bernstein, 2003; Qi et al., 2007), in Chapter titled "XML Data Integration: Merging, Query Processing and Conflict Resolution", , where we discuss integration based on mappings, we refer to a more general definition of mappings.

Matching Operation: Identifying Mappings

Matching is challenging (Gal, 2006) due to several reasons. First, identical concepts may be named or structured differently. Second, the same or similar words may be used to represent different concepts. To match two sources, one can leverage different types of cues (E. Rahm et al., 2004): (i) schema information such as data types, element names, or structures, (ii) external information such as thesauri, (iii) data instance characteristics, and (iv) previous matching results. Based on how they leverage these, it is possible to classify the available matching techniques using three broad criteria (E. Rahm & Bernstein, 2001; Shvaiko & Euzenat, 2005):

- Element-level vs. structure-level: We can classify matching algorithms based on whether the structural relationship among elements are used or not. Element-level algorithms only analyze elements themselves, but ignoring relationships among them. In contrast, structure-level matching algorithms match elements based on how they are related to each other in the overall structure.
- Instance-based vs. schema-based: The former considers data instances while resolving mappings among schema elements, while the latter only considers schemas during the matching process.
- Syntax-based vs. semantic-based: The syntax-based (or syntactic) approaches only consider syntactic cues (e.g., available thesauri), while the semantic-based methods also leverage available semantics (e.g., integrity constraints).

String-distance based (Cohen, Ravikumar, & Fienberg, 2003; Do, Melnik, & Rahm, 2002; Madhavan et al., 2001; Melnik, Garcia-Molina, & Rahm, 2002; Noy & Musen, 2001), linguistic resource based (Bouquet, Serafini, & Zanobini, 2003; Giunchiglia, Shvaiko, & Yatskevich, 2004;

Madhavan et al., 2001; G. A. Miller, 1995; Resnik, 1995), and constraint based (E. Rahm & Bernstein, 2001; Valtchev & Euzenat, 1997) approaches to matching are not specific to XML matching. Thus, we will not focus on them here. Instead, we will focus on techniques that leverage the structure of XML data and schemas as well as *hybrid* and *composite* matching approaches which use multiple (e.g., structure and semantic) techniques.

Structure-based Techniques

As discussed earlier, the tree-like structure of XML data and schemas renders the structure-based matching techniques play a fundamental role in XML schema matching. Existing structure-based techniques match elements in trees (or graphs) by either computing their similarity in the initial tree structure (Do & Rahm, 2002; Madhavan et al., 2001) or by mapping them to a multi-dimensional space to compute their closeness values (Candan, Kim, Liu, & Suvarna, 2006).

Cupid (Madhavan et al., 2001) is a generic matching approach that can work for both XML and relational databases. This approach considers both structural similarity and non-structural (e.g., linguistics and constraints) information in computing the similarity values of two elements. The similarity value of two elements v_i and v_j is a weighted similarity of all the above factors. In this section, we consider the structure-based bottom-up matching algorithm *TreeMatch*. Given two schema trees S_1 and S_2, the initial similarity value for each leaf element pair is initialized based on an assessment of how compatible the corresponding data types are. Then, *TreeMatch* computes the similarity value *sim* of every element pair (v_1,v_2) $(v_1 \in S_1 \wedge v_2 \in S_2)$ by traversing the two schema trees in post-order. Two cases need to be considered in this computation. In the first case, where the two elements are leaves, their similarity value is the weighted value of their structural similarity value *ssim* and the similarity value *lsim* computed considering other factors (e.g., linguistics). The second case occurs when one element is a non-leaf element. In such a case, the structural similarity of these two elements is measured as the fraction of leaf level element matches. One leaf element *matches* another if their weighted similarity value is higher than some threshold ε. Let v_1 and v_2 be two elements to be matched, *Leaves*(v_1) and *Leaves*(v_2) represent the leaf element sets in sub-trees rooted at v_1 and v_2 respectively. Let $\oplus$ be the union of these two leaf element sets (i.e., $\oplus$=*Leaves*(v_1)$\cup$*Leaves*(v_2)), V_1 represents the set of elements in *Leaves*(v_1) which matches some element in *Leaves*(v_2). Similarly, let V_2 represent the set of elements in *Leaves*(v_2) which matches some element in *Leaves*(v_1). Then, the *ssim*(v_1,v_2) is computed as $\frac{|V_1 \cup V_2|}{|\oplus|}$. From the structural similarity value, a weighted similarity value *sim*(v_1,v_2) over the two elements is computed. Next, this similarity value of two elements is further propagated to the leaf-element pairs in their subtrees. In particular, given two thresholds ε_h and ε_l, when *sim*(v_1,v_2)>ε_h, the structural similarity value of every leaf element pair is increased by a factor f_{inc}. On the contrary, when *sim*(v_1,v_2)<ε_l, *ssim*(v_1,v_2) is decreased by a factor f_{dec}. This process continues until all the element pairs from both trees are traversed.

Figure 5
[Put figure fig5a.tif here]
[Figure 5a subtitle: (a) Part of source schema S_1]
[Put figure fig5b.tif here]
[Figure 5b subtitle: (b) Part of source schema S_2]
[Figure 5 caption: Two source schemas]

We can use the two source schemas in Figure 5 to illustrate this process.

In this figure, to distinguish the different element names in different contexts, we associate with each one a number to make the description easier. Let the matching threshold be $\varepsilon = 0.3$. Let us also assume that initially from the data type compatible matrix, we get that *ssim*(*name2*,*gname*)=0.5, *ssim*(*name1*,*sponsor*)=0.5, *ssim*(*name3*,*gname*)=0.5, …. When we compute *ssim*(*fund*, *grant2*), we have that element *name2*∈*Leaves*(*fund*) matches *gname*∈*Leaves*(*grant2*), thus,

$$ssim(fund, grant2) = \frac{|\{name2\} \cup \{gname\}|}{|\{name2, sponsor, gname, amount\}|} = \frac{2}{4} = 0.5$$

Next, *ssim*(*fund*,*grant2*) is adjusted to its weighted score *sim*(*fund*,*grant2*). If *sim*(*fund*,*grant2*) is bigger than ε_h, then *ssim*(*name2*,*sponsor*)=f_{inc}×*ssim*(*name2*,*sponsor*), *ssim*(*name2*,*gname*)= f_{inc}×*ssim*(*name2*,*gname*), and *ssim*(*name2*,*amount*= f_{inc}×*ssim*(*name2*,*amount*).

Milo & Zohar (1998) also use schema graphs for matching; matching is performed node by node starting at the "roots" of the tree-like schema graph. More generally, let *T*(*V*,*E*) be a tree schema. *T* is called a rooted tree if one of the vertices/nodes is distinguished and called the *root*. *T* is called a node labeled tree if each node in *V* is assigned a symbol from an alphabet Σ. *T* is called an ordered tree if it is rooted and the order among siblings (nodes under the same parent node) is also given. An unordered tree is simply a rooted tree. Given two ordered labeled trees, T_1 and T_2, T_1 is said to *match* T_2 if there is a one-to-one mapping from the nodes of T_1 to the nodes of T_2 such that (a) the roots map to each other, (b) if a tree node v_i maps another tree node v_j, then the children of v_i and v_j map to each other in left-to-right order and (c) label of v_i is equal to the label of v_j. Note that exact matching can be checked in linear time on ordered trees. T_1 is said to *match* T_2 *at node v* if there is a one-to-one mapping from the nodes of T_1 to the nodes of the subtree of T_2 rooted at *v*. The naive algorithm (which checks for all possible nodes of T_2) takes $O(nm)$ time where *n* is the size of the T_1 and *m* is the size of T_2, while there are $O(n\sqrt{m})$ algorithms which leverage special index structures, such as suffix trees for compressed representation and quick access to sub-paths of T_1. While the matching problem is relatively efficient for ordered trees, the problem quickly becomes intractable for unordered trees. In fact, for unordered trees, the matching problem is known to be NP-hard (Kilpeläinen & Mannila, 1995).

As opposed to these potentially expensive approaches, Candan et al. present two approaches in (Candan et al., 2006) and in (Candan, Kim, Liu, & Agarwal, 2007), respectively that use both data instances and hierarchical structures to match two tree-structured schemas, S_1 and S_2. These methods both map the nodes of the tree into a multi-dimensional space (using multi-dimensional scaling in (J. Kruskal, 1964; J. B. Kruskal & Wish, 1978) and using propagation in (Kim & Candan, 2006) and compute the similarity values of the elements in this multi-dimensional space. These approaches work in three steps. First, the nodes in the trees are mapped into two *k*-dimensional spaces $\mathbb{R}^k_1$ and $\mathbb{R}^k_2$, respectively. Then, these spaces are aligned based on common nodes in the two trees. Finally, once the nodes from both trees are mapped onto a common space, the algorithms use clustering or nearest-neighbor algorithms to find potentially related nodes.

Hybrid and Composite Methods

The difficulty of the matching problem and the different aspects of the data and schema that can be used as cues make pure matching solutions (e.g., solely based on instance, or on structure) inadequate. Due to this, hybrid matching approaches that incorporate multiple information in

matching tend to be more effective. Different from the hybrid matching method, which utilizes diverse information but still works as one matcher, the composite matching approaches combine the results of several matching operators. In what follows, we briefly describe some well recognized hybrid and composite methods which work for XML databases.

Onion (Mitra, Wiederhold, & Kersten, 2000) and its predecessor SKAT (Mitra, Wiederhold, & Jannink, 1999) are schema-based matching systems that first perform a linguistic matching and then apply structure-based matching. The structure-based phase, which attempts to match only the unmatched terms, is based on structural isomorphism detection between the subgraphs. Clio (Hernández, Miller, & Haas, 2001; R. J. Miller, Haas, & Hernández, 2000)is a mixed schema-based and instance-based system that proposes a declarative approach to schema matching between either XML or/and relational schemas. After the first phase in which input schemas are translated into an internal representation, the system combines sequentially instance-based attribute classifications (by using a Bayes classifier) with a string matching between elements names (these *n*-to-*m* value correspondences can be also entered by the user through a graphical user interface). After that, Clio produces a final mapping. Cupid (Madhavan et al., 2001) also exploits not only structural information, but combines multiple techniques, including linguistic-based, element-based, structure-based, context-dependent matching. It also leverages internal structure, similarity of atomic elements, and constraints. In general, to compute the similarity coefficients between elements from two schemas, first, a linguistic matching is performed to match elements based on their names, data types, domains, etc. In this step, a thesaurus is also used to identify synonyms and acronyms. Then, a structural matching is run to match elements based on the similarity of their contexts or vicinities as described earlier.

Next, a final score measuring the similarity of two elements is calculated by combining the two scores obtained in the previous two steps. Finally, the mapping is deduced from these coefficients. In particular, if the adjusted similarity value *sim* of two elements v_i and v_j is no smaller (i.e., equal or bigger) than the given threshold, a correspondence from v_i to v_j (i.e., $\mu=v_i \rightarrow v_j$ (*sim*)) is generated. In the simplest case, only the leaf-level correspondences are returned. The mapping in a general case is one-to-many since a source element may map to many target elements. SF (Melnik et al., 2002) also uses a hybrid combination of name matchers and SemInt (Li & Clifton, 1994; Li & Clifton, 2000) is a hybrid approach exploiting both schema and instance information. Since they don't specifically deal with XML documents, we omit their descriptions here.

COMA (Do & Rahm, 2002) is a composite matching system working for XML database. Besides using different matching operators, COMA also reuses previous matching results. We briefly introduce the basic ideas of this system:

1. Given two schemas S_i and S_j, COMA finds a set of intermediate schemas, S_k-s, that have some matching results with S_i and S_j: $S_i \leftrightarrow S_k$ (mapping between S_i and S_k) and $S_j \leftrightarrow S_k$.
2. This step computes $S_i \leftrightarrow S_j$ (i.e., mapping between S_i and S_i) by using the previous matching results of other schemas with S_i and S_j (maybe generated by other matchers). Specifically, given l intermediate schemas, for each intermediate schema S_k, COMA uses a *MatchCompose* operation to compute $S_i \leftrightarrow S_j$ from $S_i \leftrightarrow S_k$ and $S_j \leftrightarrow S_k$. The result of *MatchCompose* process using one intermediate schema is a similarity matrix where the similarity value at the i-th row and j-th column is the similarity value for matching $s_i \in S_i$ to $s_j \in S_j$. l is the number of intermediate schemas. The combined result of *MatchCompose* process for all intermediate schema S_k-s is a $m \times n \times l$ similarity cube where each matrix is for one intermediate schema.

3. Next, for each element pair (v_i, v_j) where $v_i \in S_i$, $v_j \in S_j$, COMA computes their similarity value by aggregating the l similarity values in the similarity cube. This gets a $m \times n$ matrix. The value at the i-th row and j-th column is the aggregated similarity value derived from the matching results (computed using other matchers) with l other schemas.
4. For the elements in one schema, COMA selects its mapping candidates from the other schema using this matrix. The matrix computed in the previous step might imply that one element in a schema may match to many elements in another schema. In this step, COMA finds the best match candidate for each element.

Figure 6
[Put figure fig6.tif here]
[Figure caption: Intended matching results and automatically discovered matching results] (Adapted from (Do et al., 2002))

Measuring the Matching Quality

The diversity of the available matching algorithms necessitate objective mechanisms to compare performances of different matching algorithms. Recently, for example, Duchateau et al. (2007) proposed a benchmark to compare the quality of different matching tools. This proposal and others all rely on statistical measures comparing the degrees of false positives (wrongly identified matches) and false negatives (missed matches) against degrees of true positives (correctly identified matches) and true negatives (correctly excluded matches).

Let, as shown in Figure 6, $M=\{(vs_1,vt_1), (vs_2,vt_2),\ldots, (vs_n,\mathrm{vt}_n)\}$ denote the matching results returned by a matching algorithm and $M'=\{(vs_1',vt_1'), (vs_2',vt_2'),\ldots, (vs_m',vt_m')\}$ denote the intended matching results (i.e., ground truth). Let the number of correct correspondences be denoted as $c=|M \cap M'|$. Matching accuracy can be quantified by various measures borrowed from the information retrieval field (Do et al., 2002). These include, $precision = \frac{c}{n}, recall = \frac{c}{m}, F-measure = \frac{2 \times precision \times recall}{precision + recall}$. SemInt (Li & Clifton, 2000), for example, uses all three of these measures to evaluate their matching method. *Recall* is used in evaluating the accuracy of LSD (Doan et al., 2001). These standard measures are also used in ontology matching, e.g., QOM (Ehrig & Staab, 2004).

An alternative to *F-measure*, which combines precision and recall, is so called, *overall* measure first proposed in SF (Melnik et al., 2002), and used in COMA (Do & Rahm, 2002). The *overall* measure intends to quantify the user effort that is needed to transform a system returned matching result into the intended one. Given M, M', and c as before, the number of wrongly suggested (false positive) correspondences is (n-c) and the number of missing (false negative) correspondences is (m-c). In total, the amount of corrections (by either deleting false positive results or adding false negative results) that a user has to make is $\frac{(n-c)+(m-c)}{m}$. Based on this observation, the overall accuracy can be defined as $overall = 1 - \frac{(n-c)+(m-c)}{m}$ (note that this measure can have non-positive values). It is easy to see that $overall = recall - \frac{n-c}{m}$; i.e., *overall* refines *recall* by deducting the percentage of the wrongly suggested correspondences. Comparisons between *overall* and *F-Measure* show that, for the same precision and recall values, *overall* tends to provide more pessimistic assessments of the matching quality (Do et al., 2002).

Mapping Rule Generation

The correspondences between source and target elements are inherently ambiguous because they do not contain information on how these elements are interpreted in their own schemas, including the contexts and the referential constraints. Therefore, these simple correspondences are not always adequate to retrieve data through one integrated schema from data instances following other schemas, whereas such data retrieval operation is common in applications like data translation, data exchange, or query processing over integrated schema. To make such data retrieval operations possible, more informed mapping rules (assertions) that can semantically connect the elements in one schema to elements in others are needed. More specifically, for a given element $v \in S_1$, an assertion specifies how the instances of v should be translated to instances in S_2. The rule is generally represented as a query q_{S2} in schema S_2 in some specific language, e.g, SQL, XQuery, or XSLT. The process to generate such assertions is called *mapping rule generation*.

In data exchange or data translation, the operation of mapping rule generation happens after the correspondences are identified. The generated rules are used to interpret the data following one schema to comply with another one. In data integration, the mapping rules are generated after schema integration, and are further used during query processing (chapter titled "XML Data Integration: Merging, Query Processing and Conflict Resolution"). To generate mapping rules among relational schemas, there exist various well known techniques, such as source-to-target tuple-generating dependencies (source-to-target tgds (Fagin, Kolaitis, Miller, & Popa, 2005)), GAV (global-as-view (Lenzerini, 2002)), LAV (local-as-view (Lenzerini, 2002)), or GLAV (global-and-local-as-view (Lenzerini, 2002)) assertions. However, the direct application of these techniques to XML database is not trivial due to the hierarchical and nested structures in XML.

Attempts to automatically generate mapping rules for XML schemas include (Fuxman et al., 2006; Hernández et al., 2007; Hernández et al., 2008; Popa et al., 2002; Yu & Popa, 2004) Pankowski et al. (2007) discuss generation of XML schema mapping rules in the presence of key constraints and value dependencies. Atay et al. (2007) present approaches to generate XML to SQL mapping rules for recursive XML schemas. Kementsietsidis, Arenas, & Miller (2003) propose a language which allows specification of alternative semantics for mapping tables and shows that a constraint-based treatment of mappings can lead to efficient mechanisms for inferring new mappings. Arenas & Libkin (2005) propose to use DTDs and source-to-target dependencies together in data translation. Popa et al. (2002) present a *semantic translation* approach to generate mapping rules between hierarchical schemas from simple correspondences between simple elements. In this semantic translation, source-to-target (s-t) dependencies are generated to associate elements in the source schema to elements in the target schema in a certain way by incorporating the semantic constraints in each schema to the element correspondences. The algorithm first computes the primary paths and logical relations for the source and target schemas separately. The primary paths from a nested schema (e.g., XML schema) is a set of elements found on the paths from the root to any non-root element. E.g., $o \in org$, $no \in org.name$, $e \in edu$, $ne \in edu.name$ are all primary paths for the nested schema in Figure 2(b). Then, by taking the referential constraints among elements, the primary paths are combined to logical relations. Each logical relation is of the form "***select * from Ps where Conditions***", where *P*s are the primary paths, and *Conditions* are some equality conditions relating two elements. Then, based on the logical relations from the source and target schema, s-t dependencies are generated. A s-t dependency is in the form of "***for A exists B where C***" where *A* and *B* are logical relations in the

source and target schemas respectively, and *C* consists of the equality conditions of the subset of correspondences. To further improve the expressive power of the mapping rules, and the translation/integration performance, Fuxman *et al.* in (2006) extended (Popa et al., 2002) to generate *nested* mapping rules, which allow nesting and correlation of mappings.

Other Considerations

Piazza (A. Y. Halevy, Ives, Suciu, & Tatarinov, 2003), HepToX (Bonifati, Chang, Ho, Lakshmanan, & Pottinger, 2005), QUEST(Qi, Candan, Sapino, & Kintigh, 2006), and FICSR (Candan et al., 2008; Qi et al., 2007) recognize that it is unrealistic to expect an independent data source entering information exchange to agree to a global mediated schema or to perform heavyweight operations to map its schema to every other schema in the group. Piazza presents a mediation language for mapping both the domain and document structures and focuses on *certain answers* that hold for every consistent instance. HepToX, on the other hand, focuses on automated mapping rule generation, without explicitly considering conflicts. FICSR (Candan et al., 2008; Qi et al., 2007) uses a feedback process to incrementally improve mappings through users feedback provided within the context of queries they pose. *Pay-as-you-go* systems (Dong, Halevy, & Yu, 2007; Jeffery, Franklin, & Halevy, 2008; Sarma, Dong, & Halevy, 2008) consider probabilistic mappings, which may improve over time with new evidence, as a way of relaxing the need for enforcing full-, consistent-integration. The idea of applying user feedback to the data integration is not new. In particular, several works (Doan et al., 2001; Jeffery et al., 2008; Wu, Yu, Doan, & Meng, 2004) explore the role of user feedback in schema matching. TRIO (Benjelloun, Sarma, Halevy, & Widom, 2006) also represents alternatives probabilistically and relies on lineage information for query processing: the lineage information provides the context in which the validity of the various statements about the data and metadata can be assessed.

FUTURE RESEARCH DIRECTIONS

As discussed above, decades of efforts on schema extraction, mapping, and merging have produced a lot of promising techniques. However, unfortunately for the users of these techniques, there is still a lot of room for improvements. In what follows, we outline several trends that deserve attention.

In this section, we observed that there are many approaches to XML schema matching and integration. Different algorithms use different kinds of information. Thus, often times, they also report results on different testing data sets. This state of affairs raises some critical questions: “how solid are these algorithms?” and “how can one fairly measure the soundness of these techniques?” Gal pointed in (Gal, 2007) that all participating matchers in a benchmark test reported very poor results with only 30-40% precision and even worse, 13-45% recall. On the other hand, “even with such low precision and recall, is it fair to state that these techniques are useless?” “How useful are these matching algorithms after decades of efforts?” Thus benchmarking is a critical research direction in this domain. Recently, there are several efforts (Duchateau et al., 2007) for developing measuring for measuring the qualities of matching algorithms. At this point, however, there are no benchmarks for measuring the effectiveness of data mergining. Moreover, even when such benchmarks exist, they end up relying on statistical measures (like precision and recall), instead of measuring how useful these systems really are in helping problem solving and decision making. Our community has to answer some tough

questions about how to measure the quality of integrated data and how to develop benchmarks that measure the utility of the various algorithms to the end-user.

We also note that most recent applications abhor pre-integration of data and, instead, demand runtime (on-line) integration (E. Rahm, Thor, & Aumueller, 2007). For example, many mashup applications integrate web contents and services on demand based on specific user's input (personalized integration) or the context (context-aware integration). Moreover, some of the new data management frameworks (such as data spaces) assume very limited schema information and are based on very loosely structured data. In such highly dynamic and loosely-structured environments, traditional algorithms may not be efficient. Dynamic (or on-the-fly) data/metadata matching, "pay-as-you-go" integration are some of proposed solutions to address this challenge. In Chapter titled "XML Data Integration: Merging, Query Processing and Conflict Resolution" we will further discuss these problems and solutions.

CONCLUSION

As shown in Figure 1, XML data integration is a multi-stage process. In this chapter, we focused on the techniques for schema extraction and mapping. We note, however, these are just the starting steps of XML data integration. In order to integrate the data, we need to further perform data/metadata merging. Based on 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. We will discuss merging, query processing over integrated XML data, and cover strategies that can be used for resolving conflicts in a separate chapter, titled "XML Data Integration: Merging, Query Processing and Conflict Resolution".

References

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

Anand, A., & Chawathe, S. S. (2004). Cooperative data dissemination in a serverless environment. *CS-TR-4562, University of Maryland, College Park,*

Arenas, M., & Libkin, L. (2005). XML data exchange: Consistency and query answering. *PODS,* 13-24.

Atay, M., Chebotko, A., Lu, S., & Fotouhi, F. (2007). XML-to-SQL query mapping in the presence of multi-valued schema mappings and recursive XML schemas. *DEXA,* 603-616.

Aulbach, S., Grust, T., Jacobs, D., & Rittinger, A. K. a. J. (2008). Multi-tenant databases for software as a service: Schema-mapping techniques. *SIGMOD Conference,* 1195-1206.

Balmin, A., Papakonstantinou, Y., & Vianu, V. (2004). Incremental validation of XML documents. *ACM Trans.Database Syst., 29*(4), 710-751.

Barbosa, D., Mignet, L., & Veltri, P. (2005). Studying the XML web: Gathering statistics from an XML sample. *World Wide Web, 8*(4), 413-438.

Barbosa, D., Mignet, L., & Veltri, P. (2006). Studying the XML web: Gathering statistics from an XML sample. *World Wide Web, 9*(2), 187-212.

Benjelloun, O., Sarma, A. D., Halevy, A. Y., & Widom, J. (2006). ULDBs: Databases with uncertainty and lineage. *VLDB,* 953-964.

Bertino, E., & Ferrari, E. (2001). XML and data integration. *IEEE Internet Computing, 5(6)*, 75-76.

Bex, G. J., Martens, W., Neven, F., & Schwentick, T. (2005). Expressiveness of XSDs: From practice to theory, there and back again. *WWW,* 712-721.

Bex, G. J., Neven, F., & Bussche, J. V. d. (2004). DTDs versus XML schema: A practical study. *WebDB,* 79-84.

Bex, G. J., Neven, F., Schwentick, T., & Tuyls, K. (2006). Inference of concise DTDs from XML data. *VLDB,* 115-126.

Bex, G. J., Neven, F., & Vansummeren, S. (2007). Inferring XML schema definitions from XML data. *VLDB,* 998-1009.

Bex, G. J., Neven, F., & Vansummeren, S. (2008). SchemaScope: A system for inferring and cleaning XML schemas. *SIGMOD Conference,* 1259-1262.

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.

Bouquet, P., Serafini, L., & Zanobini, S. (2003). Semantic coordination: A new approach and an application. *International Semantic Web Conference,* 130-145.

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., Kim, J. W., Liu, H., & Agarwal, R. S. a. N. (2007). Multimedia data mining and knowledge discovery. (pp. 259-290) Springer London.

Candan, K. S., Kim, J. W., Liu, H., & Suvarna, R. (2006). Discovering mappings in hierarchical data from multiple sources using the inherent structure. *Knowl.Inf.Syst., 10*(2), 185-210.

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.

Chidlovskii, B. (2001). Schema extraction from XML: A grammatical inference approach. *KRDB,*

Chitic, C., & Rosu, D. (2004). On validation of XML streams using finite state machines. *WebDB,* 85-90.

Chung, C., Min, J., & Shim, K. (2002). APEX: An adaptive path index for XML data. *SIGMOD Conference,* 121-132.

Clark, J. *Trang: Multi-format schema converter based on RELAX NG*

Cohen, W. W., Ravikumar, P., & Fienberg, S. E. (2003). A comparison of string distance metrics for name-matching tasks. *IIWeb,* 73-78.

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.

Devlin, B. A., & Murphy, P. T. (1988). An architecture for a business and information system. *IBM Systems Journal, 27*(1), 60-80.

Do, H. H., Melnik, S., & Rahm, E. (2002). Comparison of schema matching evaluations. *Web, Web-Services, and Database Systems,* 221-237.

Do, H. H., & Rahm, E. (2002). COMA - A system for flexible combination of schema matching approaches. *VLDB,* 610-621.

Doan, A., Domingos, P., & Halevy, A. Y. (2001). Reconciling schemas of disparate data sources: A machine-learning approach. *SIGMOD Conference,* 509-520.

Doan, A., & Halevy, A. Y. (2005). Semantic integration research in the database community: A brief survey. *AI Magazine, 26*(1), 83-94.

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

Duchateau, F., Bellahsene, Z., & Hunt, E. (2007). XBenchMatch: A benchmark for XML schema matching tools. *VLDB,* 1318-1321.

Ehrenfeucht, A., & Zeiger, H. P. (1976). Complexity measures for regular expressions. *J.Comput.Syst.Sci., 12*(2), 134-146.

Ehrig, M., & Staab, S. (2004). QOM - quick ontology mapping. *International Semantic Web Conference,* 683-697.

Fagin, R., Kolaitis, P. G., Miller, R. J., & Popa, L. (2005). Data exchange: Semantics and query answering. *Theor.Comput.Sci., 336*(1), 89-124.

Fernau, H. (2004). Extracting minimum length document type definitions is NP-hard. *ICGI,* 277-278.

Fernau, H. (2005). Algorithms for learning regular expressions. *ALT,* 297-311.

Florescu, D. (2005). Managing semi-structured data. *ACM Queue, 3(8)*, 18-24.

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.

Fuxman, A., Hernández, M. A., Ho, C. T. H., Miller, R. J., Papotti, P., & Popa, L. (2006). Nested mappings: Schema mapping reloaded. *VLDB,* 67-78.

Gal, A. (2006). Why is schema matching tough and what can we do about it? *SIGMOD Record, 35*(4), 2-5.

Gal, A. (2007). The generation Y of XML schema matching panel description. *XSym,* 137-139.

Garcia, P., & Vidal, E. (1990). Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Trans.Pattern Anal.Mach.Intell., 12*(9), 920-925.

Garofalakis, M. N., Gionis, A., Rastogi, R., Seshadri, S., & Shim, K. (2003). XTRACT: Learning document type descriptors from XML document collections. *Data Min.Knowl.Discov., 7*(1), 23-56.

Giunchiglia, F., Shvaiko, P., & Yatskevich, M. (2004). S-match: An algorithm and an implementation of semantic matching. *ESWS,* 61-75.

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

Gottlob, G., Koch, C., Pichler, R., & Segoufin, L. (2005). The complexity of XPath query evaluation and XML typing. *J.ACM, 52*(2), 284-335.

Halevy, A. (1999). More on data management for XML. *White Paper, Available Online at: Http://www.Cs.Washington.edu/homes/alon/widom-Response.Html,*

Halevy, A. Y., Franklin, M. J., & Maier, D. (2006). Principles of dataspace systems. *PODS,* 1-9.

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.

Hegewald, J., Naumann, F., & Weis, M. (2006). XStruct: Efficient schema extraction from multiple and large XML documents. *ICDE Workshops,* 81.

Hernández, M. A., Ho, H., Popa, L., Fuxman, A., Miller, R. J., Fukuda, T., et al. (2007). Creating nested mappings with clio. *ICDE,* 1487-1488.

Hernández, M. A., Miller, R. J., & Haas, L. M. (2001). Clio: A semi-automatic tool for schema mapping. *SIGMOD Conference,* 607.

Hernández, M. A., Papotti, P., & Tan, W. C. (2008). Data exchange with data-metadata translations. *PVLDB, 1*(1), 260-273.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to automata theory, languages and computation* Addison-Wesley Publishing Company.

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.

Jennifer, R. G., & Widom, J. (1999). Approximate DataGuides. *In Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats,* 436-445.

Jhingran, A. (2006). Enterprise information mashups: Integrating information, simply. *VLDB,* 3-4.

Kementsietsidis, A., Arenas, M., & Miller, R. J. (2003). Mapping data in peer-to-peer systems: Semantics and algorithmic issues. *SIGMOD Conference,* 325-336.

Kilpeläinen, P., & Mannila, H. (1995). Ordered and unordered tree inclusion. *SIAM J.Comput., 24*(2), 340-356.

Kim, J. W., & Candan, K. S. (2006). CP/CV: Concept similarity mining without frequency information from domain describing taxonomies. *CIKM,* 483-492.

Kintigh, K. W. (2006). The promise and challenge of archaeological data integration. *American Antiquity,*

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

Kruskal, J. (1964). Nonmetric multidimensional scaling: A numerical method. *Psychometrika, 29*(2), 115-129.

Kruskal, J. B., & Wish, M. (1978). *Multidimensional scaling* SAGE publications, Beverly Hills.

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.

Li, W., & Clifton, C. (1994). Semantic integration in heterogeneous databases using neural networks. *VLDB,* 1-12.

Li, W., & Clifton, C. (2000). SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data Knowl.Eng., 33*(1), 49-84.

Madhavan, J., Bernstein, P. A., & Rahm, E. (2001). Generic schema matching with cupid. *VLDB,* 49-58.

Martens, W., Neven, F., Schwentick, T., & Bex, G. J. (2006). Expressiveness and complexity of XML schema. *ACM Trans.Database Syst., 31*(3), 770-813.

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.

Melnik, S., Garcia-Molina, H., & Rahm, E. (2002). Similarity flooding: A versatile graph matching algorithm and its application to schema matching. *ICDE,* 117-128.

Miller, G. A. (1995). WordNet: A lexical database for english. *, 38(11)* 39-41.

Miller, R. J., Haas, L. M., & Hernández, M. A. (2000). Schema mapping as query discovery. *VLDB,* 77-88.

Milo, T., & Suciu, D. (1999). Index structures for path expressions. *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings; Lecture Notes in Computer Science, , 1540* 277-295.

Milo, T., & Zohar, S. (1998). Using schema matching to simplify heterogeneous data translation. *VLDB,* 122-133.

Min, J., Ahn, J., & Chung, C. (2003). Efficient extraction of schemas for XML documents. *Inf.Process.Lett., 85*(1), 7-12.

Mitra, P., Wiederhold, G., & Jannink, J. (1999). Semi-automatic integration of knowledge sources. *Proc. 2nd International Conference on Information Fusion,* 572–581.

Mitra, P., Wiederhold, G., & Kersten, M. L. (2000). A graph-oriented model for articulation of ontology interdependencies. *EDBT,* 86-100.

Noy, N., & Musen, M. (2001). Anchor-PROMPT: Using non-local context for semantic matching. *Proceedings of the Workshop on Ontologies and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAI),* 63-70.

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

Pankowski, T., Cybulka, J., & Meissner, A. (2007). XML schema mappings in the presence of key constraints and value dependencies. *EROW,*

Papakonstantinou, Y., Garcia-Molina, H., & Widom, J. (1995). Object exchange across heterogeneous information sources. *ICDE,* 251-260.

Popa, L., Velegrakis, Y., Miller, R. J., Hernández, M. A., & Fagin, R. (2002). Translating web data. *VLDB,* 598-609.

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

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

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,*

Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *VLDB Journal, 10(4)*, 334-350.

Rahm, E., Do, H. H., & Massmann, S. (2004). Matching large XML schemas. *SIGMOD Record, 33*(4), 26-31.

Rahm, E., Thor, A., & Aumueller, D. (2007). Dynamic fusion of web data. *XSym,* 14-16.

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. *IJCAI,* 448-453.

Sankey, J., & Wong, R. K. (2001). Structural inference for semistructured data. *CIKM,* 159-166.

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.

Segoufin, L., & Vianu, V. (2002). Validating streaming XML documents. *PODS,* 53-64.

SGML. *Standard generalized markup language: Http://www.w3.org/MarkUp/SGML/*

Shvaiko, P., & Euzenat, J. (2005). A survey of schema-based matching approaches. *J.Data Semantics IV,* , 146-171.

Shvaiko, P., & Euzenat, J. (2008). Ten challenges for ontology matching. *OTM Conferences (2),* 1164-1182.

Valtchev, P., & Euzenat, J. (1997). Dissimilarity measure for collections of objects and values. *IDA,* 259-272.

Wu, W., Yu, C., Doan, A., & Meng, W. (2004). An interactive clustering-based approach to integrating source query interfaces on the deep web. *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data,* Paris, France. 95-106.

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

XMLschema. *Http://www.w3.org/XML/Schema*

Yu, C., & Popa, L. (2004). Constraint-based XML query rewriting for data integration. *SIGMOD,* 371.