Schema Matching for Document Exchange:
A Constraint Based Approach

Sana Sellami 1 Aicha-Nabila Benharkat 1 Rami Rifaieh 2 Youssef Amghar 1

1 LIRIS, National Institute of Applied Science of Lyon, Villeurbanne, France
2 San Diego Supercomputer Center, University of California San Diego, San Diego, USA
rrifaieh@sdsc.edu

Abstract. Information systems’ activities are increasingly becoming distributed. Many systems, therefore, need to exchange documents in order to correctly perform a critical activity. Data stores on the originating and receiving systems rarely have identical structures. As a consequence, document exchange commonly requires that the data be transformed from the originating site, to the document exchange format, and again to the receiving site format. A growing number of document exchange behavior creates new requirements, such as automatically matching document structure, simplifying user’s tasks in creating mappings, and automatically generating data transformation. In our work, we are interested in all these processes: matching, mapping, and data transformation. We propose, in this paper, to extend the XML schema matching used for document exchange to cover constraints management. We show how the constraints improve the performance of schema matching. We also propose XME (XML Mapping Expression) which is an expression model of mappings along with a collection of transformation operators. These extensions take part in our platform for document exchange (ASMADE) that automates matching and generates data transformation instances.

Key words: Integration, XML schemas, Constraints, Schema Matching, Schema Mapping, Mapping Expression Model.

1. Introduction

The current informational environment is characterized by strongly distributed systems. Reducing back-office costs and enhancing the speed and effectiveness of these systems can be reached with message based data exchange. These data are wrapped in a well formatted exchanged document. For instance, health systems follow HIPAA (Health Insurance Portability and Accountability Act) [19] to offer a simplified way for managing health financial claims and reimbursement through EDI (Electronic Data Interchange) standards.

Data interchange standards specify structure and content of messages or data sets being exchanged between systems [24]. However, data stores on the originating and receiving systems rarely have identical structure. As consequence, data interchange commonly requires that the data be transformed twice: once when the message is assembled at the originating site, and again when it is parsed at the receiving site [21].
In order to optimize the transformation process, schema matching was suggested in [2] [20] for XML based EDI. In this respect, schema matching identifies a set of similarity between the components of two document/data structure. Furthermore, a unified platform called ASMADE (Automated Schema Matching for Documents Exchange) was suggested in [17] for handling matching, mapping and code generation.

We are interested, in this paper, in extending ASMADE platform. Firstly, we suggest a new dimension for schema matching that includes constraints. The constraints are essential source of semantic integrity for exchanged data. In XML based document exchange model, these constraints can be defined using XML Schema constraints on the structure and content of the documents. The constraints in a schema may also include data type assignments that affect how information is processed.

Secondly, we define the process of mapping XML schemas and we propose XME (XML Mapping Expression) which is an expression model for mappings. This model defines theoretically usable transformations rules following the matchmaking generated by the process of matching. We describe over this model a collection of transformation operators to cover useful cases for the generation of target elements.

This paper is organized in the following way. In section 2, we discuss related works concerning matching and mapping processes as well as the constraints managements. In section 3, we present ASMADE architecture. In section 4, we propose the extension of EXSMAL (matching and filtering layer) to constraints. In Section 5, we describe our proposed XML Mapping Expression (XME). Section 6 presents a prototype of this platform. Finally, we conclude the paper with the discussion and future work.

2. Related works

The automation of the data integration and schema matching has been studied for a long time. We can distinguish between two categories of approaches: matching and mapping.

2.1 Management of constraints in schema Matching

The Matching is an operation that takes in input two data schemas and returns the semantic similarities values of their elements. Several works have been achieved in order to provide matchmaking algorithms that manage the correspondences or the incompatibilities of the schema. We are especially interested in matching algorithms that manage constraints. In the literature, Cupid [3] treats the constraints of schemas as the types of data and the ranks of values, the cardinality, etc. Similarity Flooding [4] [5], structural Matching algorithm for XML schemas, SQL DDLS, RDF schemas, UML and OEM, treats the constraints that are based on the primary keys, unique keys or referential constraints that refer to the foreign and constrained keys of cardinality. XClust [6] treats the constraints of cardinality that can be imposed on the elements of the DTD as well as the constraints of similarity, compatibility. In the setting of our approach, we will work more precisely on the EXSMAL [2] algorithm. We search
therefore to extend the EXSMAL algorithm to constraint management. This algorithm will be clarified hereafter. We study, therefore, the different constraints related to the XML schemas as described in [22], [23].

2.2 Schema Mapping

In the second category, the process of mapping consists of completing the correspondences values with the true semantic relation. Several models have been proposed for this goal, and offer operations of transformations that are given by the semantic relations such as equivalent, more general, incompatible, compose, is composed of, etc. The LIMXS model [7] (Layered Interoperability Model for XML Schemas) offers a semantic view for XML schemas through the concepts specification and semantic relationship among them. This model is based on a set of transformation operations on schemas’ entities that can be composed with each others to represent more transformations. The approach proposed in [8] is an extension of the LIMXS approach called XML Hyperschema. In Model Management System [9], the operators are generic and are applied only on the schemas (called models). The approach presented in [10] [11] creates an interactive Mapping based on correspondences values that show how the value of a target attribute can be created from a set of source attributes values. The mapping expressions are studied like values of correspondences. A Mapping Expression Model has been proposed in [1]. This model is an extension of the previous approach since it adds some conditions and defines a filter for source within mapping expression between the similar elements. In addition to all these models, several tools achieving the mapping exist on the market. We have established a comparative study in [12] between these different tools and we have identified the limits of each. We conclude that all these tools don't realize simultaneously the matching, mapping and code generation. Our purpose is to realize a platform that combines all this process to simplify the task of users working on document exchange.

3. ASMADE Platform

We present in this section the ASMADE system (Automated Schemas MAthing for Documents Exchange) (fig.1). This platform was proposed in [17] and we tend, in this paper, to extend this platform. Traditionally, ASMADE contains 4 main layers:

- **Matching Layer**: Entries (XSD schemas) are transformed into trees by the EXSMAL algorithm (matching layer). The process of Matching produces EC, a set of correspondences (1-1, 1-n, n-m). This set is verified according to the constraints expressed on the elements. EXSMAL has been proposed as a solution for the EDI message's schema matching [2]. The criteria for matching include data-type, structure, and elements descriptions. The choice for the XML schema was motivated by its potential to define the structure and the semantics of an EDI message. However, the matching in EXSMAL doesn’t take yet into account the information related to an element (constraints, status, cardinality). For this purpose, we have proposed and defined a classification of different constraints and an algorithm to take into consideration the intrinsic constraints.
Filtering Layer: A filtering layer permits to eliminate irrelevant correspondences. Herein, we applied matching constraints (intrinsic and process constraints) to prune insignificant matching in $EC_1$ (set of correspondences).

Mapping Layer: In this third layer, we use the previous result designated by $EC_2$ to generate some usable rules of transformation expressed in the XME model (XML Mapping Expression).

Layer transformation: In this last layer, the resulting mappings will be generated in XQuery and executed by XQuery engine. Each Mapping will be transformed according to XQuery’s FLWR expression (For-Let-Where-Return).

4. Extension the matching process to constraints management

4.1 Constraints matching Classification

We consider that the search of the correspondences between the XML schemas elements must benefit from various types of constraints. These constraints are an essential source of semantics for matching between elements. The use of these constraints plays a key role in pruning away weak matches. In other words, the information about elements’ constraints can be used as discriminators to determine the likelihood that two elements are equivalent. We distinguished two categories of constraints: the intrinsic constraints and the process constraints.

4.1.1 Intrinsic Constraints

The intrinsic constraints are defined and applied on the elements of the XML Schema. These constraints are generally related to the information on the schemas. This information could be the representation of the elements, their names, their textual descriptions and structures, relations between the elements, the statutes, and the cardinality of the elements. The intrinsic constraints can be classified in two categories [13] covering simple and complex type in XML schemas.
C1: Constraints on the simple types
The simple types don't contain other elements or attributes. These simple types are defined or are predefined. Their constraints are defined by the facets as the minimum, maximum value for the numbers, dates and lengths, and the patterns, the minimum, maximum length for the chains, names, ID.

Example: `<xsd:simpleType>
  <xsd:restriction base="xsd:positiveInteger">
  <xsd:maxExclusive value="100"/>
  </xsd:restriction>
</xsd:simpleType>`

C2: Constraints on the complex types
Complex types are the elements that contain other elements and/or attributes. Their constraints include:

- **Ct**: Type Constraint: These are supplementary constraints data while associating some datatypes to the XML document data. These constraints concern predefined datatypes (i.e. primitive and derived datatypes).
  Example: `<xsd:restriction base="xsd:string">
    <xsd:pattern value="[^0-9]"/>
  </xsd:restriction>`

- **Cp**: Property Constraint: An XSD attribute element can have three optional attributes: use, default and fixed. These attributes translate primary keys of which the values are not null and unique. Combinations of these three parameters define what is acceptable in the XML final document (e.g. mandatory attribute, optional, default value, etc.).
  Example: `<xsd:attribute name="partNum" marks="SKU" uses="required"/>
  <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="item" type="xsd:positiveInteger"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>`

- **Co**: Occurrence Constraint: the occurrence is determined by the attributes minOccurs that can only be a positive integer and not null and maxOccurs that can also be a positive integer and not null or unbounded. These attributes are controlled by the attribute “use” such as required, optional or prohibited values.
  Example: `<xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="item" type="xsd:positiveInteger"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>`

- **Cc**: Composition Constraint: These constraints are the constraints on the elements (i.e.: sequence, choice, all).
- **Cs**: Structural Constraint: These constraints correspond to the determination of the structure of the document XML. The structure defines especially the XML elements: name, number and the nature of their attributes as well as their content.
  Example: `<xsd:element name="element_name" type="xsd:type_name"/>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="element_name" type="xsd:type_name"/>
    </xsd:sequence>
  </xsd:complexType>
  </xsd:element>`

- **Cr**: Reference Constraint [14]: These constraints can be a source of semantics for the relations between the elements and more precisely between the attributes. This type of information can be determined through the ID attributes and IDREF.
- **Cn**: Conceptual Constraint: If two elements are matched they must be similar to the same concept. This concept is defined by user predefined ontologies.
- **Cg**: Global Constraint: These are the constraints about the elements and the global attributes that are referenced by “ref”.

4.1.2 Process Constraints
Process constraints are specified by the user or by the experts of the domain. These constraints can be added or modified according to the needs. They play a key role in the determination of semantic correspondences between the schemas in the domain and are important because they are essential part of consistency framework. If these constraints are observed during the process of matching, we can potentially improve the accurateness of the matching. Thus, we add a filter to our architecture that eliminates the matching that violates these constraints.
· Cf: The Frequency Constraint: imposes some regularities to which a source schema must be compliant. For example at most one element source matches CITY or precisely an element source matches STATE.

· Ca: The Adjacent Constraints: imply that if two elements are not related as “street and quantity” meaning that they cannot participate in the process of the matching. The user, in this case, can use this constraint for never generate a candidate to the matching that could combine street and quantity.

· Cm: The Mutual Constraint [16]: expresses for example one element e belonging to a schemas source S can matches an element targets f which belongs to a targets schema T if and only if e is one of the elements most similar to f and mutually if f is one of the elements most similar to e.

· Cu: The Uniqueness Constraint: verifies that every element must be matched with a distinct element in the schemas targets.

· Cmp: The Constraint of previous matchings [15]: takes into consideration a previous matching result, and permits to exploit it to minimize the matching operations.

4.2 Matching Intrinsic Constraints

We propose an algorithm that treats the constraints on the simple types (C1), type (Ct) and occurrence (Co) constraints.

4.2.1 Algorithm Description

Let ECj be a set of matched elements. We are going to determine for every node ei of the source schema S and node fj of the target schema T if their correspondences don't violate the intrinsic constraints. Here we wish to determine the matching which respect the constraints and filter ECj pairs with respect to the penalty violation of the constraints.

A brief description of the used methods is as follows:

**IsCompatible:** this method returns the compatibility between the constraints; it covers all the possible cases of simple and complex constraints. The compatibility is true if ei and fj have the same type of constraints. The different cases are when: ei has simple with fj has simple constraints or ei has simple with fj has simple and complex constraints (we ignore the complex and just match the simple); ei has complex with fj has complex constraints; ei has complex with fj has simple and complex constraints (ignore the simple) and finally ei has simple and complex with fj has simple and complex.

**isConstraint(ei,Ck):** returns true if Ck is an existent constraint for (ei).

**isFiniteNumericConstraint:** covers all the constraints that are specified on the numeric finite domain like C1,Ct,Co.

**ConstraintType:** This method determines the type of constraint defined on the element. It can be simple or complex type.

**ComputePenaltyViolation:** This method calculates the penalty of violation to the matching constraints. This value is calculated in function of the weight of the constraint and the value of the violation of constraint with respect to NbCommonConstraint.

**ComputeConstraintViolation:** This method calculates the violation of a constraint, we illustrate in the algorithm the method used for numeric finite domain. It uses domain intersection and the cardinality of the intersection versus the cardinality of each domain.
4.2.2 Pseudo-code

Inputs:
- \( EC_1 \): Set of Similarities between nodes (result of matching), \( EC_1 = \{ (e_i, f_j, val_1) | i=1,n, j=1,m \} \) where \( e_i \in S \) (source element) and \( f_j \in T \) (target element).
- Constraints \( C_1 \) and \( C_2 \) on elements: \( C_1 \) is the set of constraints on simple types and \( C_2 \) is the set of constraints on complex types. Thus, \( C_1 = (C_{st}, \omega_0) \) and \( C_2 = (\{Ct, \omega_1\}, \{Cp, \omega_2\}, \{Cc, \omega_3\}, \{Co, \omega_4\}, \{Cn, \omega_5\}, \{Cr, \omega_6\}, \{Cg, \omega_7\}, \{Cs, \omega_8\}) \) where \( \omega = \{ \omega_0, \ldots, \omega_8 \} \) the weight associated with each constraint.

Outputs:
The set of refined correspondences \( EC_2 \), where \( EC_2 = \{ (e_i, f_j, val_2) | i=1,n, j=1,m \} \)

\[
\text{ConstraintMatching (EC}_1, C_1, C_2) \\
\text{Begin} \\
C_{pvs}, C_{pvc} = 0 \\
EC_2 = \{\} \\
\text{For each } (e_i, f_j, val_1) \in EC_1 \\
\quad \text{If (isCompatible(ConstraintType(e_i), ConstraintType(f_j)))} \\
\quad \quad \text{Then} \\
\quad \quad \quad \text{If (isSimple(ConstraintType(e_i)))} \\
\quad \quad \quad \quad \text{Then } C_{pvs} = \text{ComputePenaltyViolation}(e_i, f_j, C_1) \\
\quad \quad \quad \quad \text{End If} \\
\quad \quad \quad \text{If (isComplex(ConstraintType(e_i)))} \\
\quad \quad \quad \quad \text{Then } C_{pvc} = \text{ComputePenaltyViolation}(e_i, f_j, C_2) \\
\quad \quad \quad \text{End If} \\
\quad \text{End If} \\
\quad \text{If } (C_{pvs} = 0 \text{ or } C_{pvc} = 0) \\
\quad \quad \text{Then } C_{m} = \text{NotZero Between}(C_{pvs}, C_{pvc}) \\
\quad \quad \text{Else } C_{m} = \text{Avg}(C_{pvs}, C_{pvc}) \\
\quad \text{End If} \\
\quad \text{If } (C_{m} \leq 0.5) \\
\quad \quad \text{Then } \text{val} = \text{val}_1 \\
\quad \quad \text{EC}_2 = \text{EC}_2 \cup (e_i, f_j, \text{val}_1) \\
\quad \text{End If} \\
\text{End For} \\
\text{Return EC}_2 \\
\text{End}
\]

\[
\text{ComputePenaltyViolation}(e_i, f_j, C) \\
\text{Begin} \\
\text{Int NbCommonConstraint} = 0 \\
C_{pv} = 0 /\text{ penalty violation} \\
\text{For each constraint } C_i \in C \\
\quad \text{If (isConstraint (e_i, C_i) and isConstraint (f_j, C_i))} \\
\quad \quad \text{Then} \\
\quad \quad \quad \text{Vc} = \text{ComputeConstraintViolation}(e_i, f_j, C_i) \\
\quad \quad \quad \text{NbCommonConstraint}++ \\
\quad \quad \quad C_{pv} = C_{pv} + \omega_i * (1 - Vc) \\
\quad \text{End If} \\
\text{End For} \\
\text{If } (\text{NbCommonConstraint} \neq 0) \\
\quad \quad \text{Then } C_{pv} = C_{pv} / \text{NbCommonConstraint} \\
\text{End If} \\
\text{Return } C_{pv} \\
\text{End}
\]

\[
\text{ComputeConstraintViolation}(e_i, f_j, C_i) \\
\text{Begin} \\
\text{If (isFiniteNumericConstraint(C_i))} \\
\quad \text{Then } //\text{look for domain compatibility and intersection}
\]
If \( \{D(e_i, C_k) \cap D(f_j, C_k) \neq \emptyset\} \)

Then If \( \{D((e_i, C_k) \cap D(f_j, C_k) = D(e_i, C_k) \) or \\
\( D((e_i, C_k) \cap D(f_j, C_k) = D(f_j, C_k) \) \\
Then \( V_{c_{i,j}} = 0 \)

Else \\
\( V_{c_{i,j}} = \text{Card(DomainIntersection}(e_i, f_j, C_k)) / (\text{Card(D}(e_i, C_k)) + \text{Card(D}(f_j, C_k))) \)
End If
End If

Else // Other Methods for non numeric constraints
// not described in this algorithm
End If
Return \( V_{c_{i,j}} \)
End

\[ 5. \text{XME: XML Mapping Expression} \]

\[ 5.1 \text{Mathematic Representation} \]

Based on the Mapping Expression Model defined in [1], we propose XME (XML Mapping Expression Model) that identifies how to generate the value of a target element/attribute from sources values. We define an attribute as follows:

\[ A_i = \sum_{j} [ f_i(a_r, a_p, ..., a_s), l_i, c_i ] + \tau_i = \alpha_i(A) + \tau_i \]

\( A_i \) represents an element/field of the target representation. \( \alpha_i(A) \) is a mapping expression applied on multi-sources to generate a part of the target field \( A \) or to generate a sub-field of the field \( A \), \( \tau_i \) is a condition on the mapped value of the field \( A \). This enables us to ignore mapped results that do not satisfy the condition. \( \tau_i \) represents a generic function, independent of the source. The generic function covers all the cases that the mapping expression model doesn’t generate. This function can be a constant value or, an incremental function, an automatic function or a skolem function. We present (in fig.2) the XME model (expressed in XSD) that defines all the properties of the XML schemas as well as the used transformation operators.

\[ 5.2 \text{transformation operations} \]

The set of primitive transformation operations are the building blocks that would enable schemas transformation. These primitive operations can be composed together to represent larger transformations. They are summarized as follows:

Add: describes an element that appears in the schemas targets but it is not in the source schema.
Remove: describes an element in the schemas source that doesn’t appear in the target schema.
Concat: describes an element of the target field that is established by concatenation of the elements of the source. (intra schema).
Split: This is the reverse operation of Concat.
Connect: Corresponds to a Matching 1-1 that joins two elements equivalent without any modification (identity).
Apply: This operator is used when we need some functions to transform the content of the source values into targets values. These functions can be by example the conversion of units or a mathematical function (min, avg, div, etc.).

Join: plays the role of an inter schema concat.

Rename: relates identical elements except that the name of the element or type that change.

Delete: to suppress the elements that doesn’t participate in the Mapping.

5.3 Example

We describe in this section how the different operators can be used. In this example (fig.3), we use the operator Join between ISBN$_{S1}$ (in the s source schema $S_1$) and ISBN$_{S2}$ (in the source schema $S_2$) for generating mappings with the element of target schema ISBN$_{TS}$.

6. Implementation

We have realized an ASMADE prototype. We integrate the EXSMAL algorithm with restructuring the code source of EXSMAL. We have realized a new Graphical user Interface with adding new functionality of editing the mapping between matched elements. We have improved the GUI by replacing the 2 panels in the interface by a canvas that accepts the drag and drop of objects representing schemas, mapping.
functions, multiple schemas, etc. This development has been realised in Java (J2EE JDK 1.5, Eclipse IDE). A snapshot of the ASMADE prototype is presented in Fig. 4.

Fig.3. Example of generated mappings between XML Schemas

![Diagram of generated mappings between XML Schemas]

Fig.4. Snapshot of the matching generated by EXSMAL

7. Conclusion and future Work

Due to the extensive use of XML in several domains as universal data exchange format, there has been a great interest on proposing rich matching and mapping processes for simplifying user’s tasks. We have realized a study on the different solutions and approaches proposed to realize the management of constraints in schema Matching as well as the mapping. After that, we have defined the different constraints that can be applied on the elements (sources and destinations) of schemas and established a classification of these constraints. We have also proposed an algorithm which takes into consideration the intrinsic constraints and returns the penalty of violation of constraints to determine the degree of similarity between the schemas. We have proposed also XME (XML Mapping Expression) which is an expression model of mappings and we described a collection of transformation operators to realize the mapping and to determine the semantic relation between schemas. All these propositions appear in the ASMADE architecture (Automated Schema Matching for Documents Exchange) that offers a framework to cope with complete approach of documents exchange. We intend in the future to improve the algorithm of constraints including all the constraints (intrinsic and process), improve the prototype of ASMADE, and integrate the algorithm of constraints. The front end of this framework is also indented to be used for ontologies and taxonomies matching in other fields such as data management in biology.
8. References

22. http://www.w3.org/TR/XMLSchema-1/