Semi-automated schema integration with SASMINT

Unal, O.; Afsarmanesh, H.

DOI
10.1007/s10115-009-0217-z

Publication date
2010

Document Version
Final published version

Published in
Knowledge and Information Systems

Knowl Inf Syst (2010) 23:99–128
DOI 10.1007/s10115-009-0217-z

REGULAR PAPER

Semi-automated schema integration with SASMINT

Ozgul Unal · Hamideh Afsarmanesh

Received: 4 August 2008 / Revised: 19 February 2009 / Accepted: 11 April 2009 /
Published online: 19 June 2009

Abstract The emergence of increasing number of collaborating organizations has made clear the need for supporting interoperability infrastructures, enabling sharing and exchange of data among organizations. Schema matching and schema integration are the crucial components of the interoperability infrastructures, and their semi-automation to interrelate or integrate heterogeneous and autonomous databases in collaborative networks is desired. The Semi-Automatic Schema Matching and INTegration (SASMINT) System introduced in this paper identifies and resolves several important syntactic, semantic, and structural conflicts among schemas of relational databases to find their likely matches automatically. Furthermore, after getting the user validation on the matched results, it proposes an integrated schema. SASMINT uses a combination of a variety of metrics and algorithms from the Natural Language Processing and Graph Theory domains for its schema matching. For the schema integration, it utilizes a number of derivation rules defined in the scope of the research work explained in this paper. Furthermore, a derivation language called SASMINT Derivation Markup Language (SDML) is defined for capturing and formulating both the results of matching and the integration that can be further used, for example for federated query processing from independent databases. In summary, the paper focuses on addressing: (1) conflicts among schemas that make automatic schema matching and integration difficult, (2) the main components of the SASMINT approach and system, (3) in-depth exploration of SDML, (4) heuristic rules designed and implemented as part of the schema integration component of the SASMINT system, and (5) experimental evaluation of SASMINT.

Keywords Collaboration · Data sharing · Heterogeneity · Schema matching · Schema integration

O. Unal (✉) · H. Afsarmanesh
Informatics Institute, University of Amsterdam,
Amsterdam, The Netherlands
e-mail: O.Unal@uva.nl

H. Afsarmanesh
e-mail: H.Afsarmanesh@uva.nl

1 Introduction

The increase in the number of databases has entailed the management of related data in different formats across these databases. In order for organizations to use other organizations' data for better decision-making and success, they need to understand the semantics and retrieve from these other distributed and heterogeneous data sources. As a result, the need for integrating or interconnecting these databases or enabling interoperability between them has become apparent. This requirement has become clearer with the increasing demand for collaboration among organizations.

More and more organizations understand the need to work together in order to better achieve their common goals. As a result of this tendency towards collaboration, during the recent years, information and communication technologies (ICT) have been focusing on how to make the collaboration among organizations easier by providing some supporting tools. In order to facilitate collaboration among organizations, one of the requirements that need to be met is enabling transparent access to heterogeneous data shared by other organizations.

A variety of approaches have been defined in research aiming at database sharing and integration, such as multidatabases and federated databases. There is however no single definition for these approaches, rather different varieties of each are addressed in publications. In this paper, we use the following definitions: A *multidatabase* system is a distributed database system, where databases can be either homogeneous or heterogeneous. According to the classification of Sheth et al. [58], there are two types of multidatabase systems: *non-federated* and *federated*. Component database systems in a *non-federated database system* are not autonomous. On the other hand, the components in a *federated database system* preserve their autonomy while also sharing their data in a partial and controlled manner. Federated databases are further categorized by Sheth et al. [58] as *loosely coupled* and *tightly coupled*. In *loosely coupled systems*, there is no single global schema and the aim is to make databases interoperable. On the other hand, in a *tightly coupled system*, there is a single global schema or multiple federated schemas if it has multiple federations. For generating both the global schema in a tightly coupled system and the federated schema(s) in a loosely coupled system, schema integration is required.

Schema integration has been a fundamental issue in data sharing among distributed, heterogeneous, and autonomous databases. With the increasing number of databases, integration problem has become more apparent. Schema integration aims at finding a unified representation of schemas by merging them. In order to integrate schemas, syntactic, semantic, and structural relationships among elements of these schemas need to be identified.

Schema integration has been the subject of intensive research in databases. Two types of schema integration are defined: (1) *View Integration*, which is performed during the database design process, for example at the conceptual design phase, and (2) *Database Integration*, which produces the global schema of a number of databases [8]. Considering the goals of our work, we focus on the *database integration*. Database integration has been an important problem that needs to be tackled within the increasing number of distributed databases.

Research on schema integration has resulted in a number of methodologies. In Batini et al. [8] several methodologies, including Batini and Lenzerini [7], Dayal and Hwang [17], ElMasri et al. [20], Mannino and Effelsberg [43], Motro and Buneman [49], are described for schema integration. In all of the cases, there is an assumption of full semantic and structural knowledge available to the integrator. Furthermore, most of the methodologies do not aim at developing semi-automated systems. However, since schema integration is a difficult and complex task, there is a need to help users with this complicated task by providing some semi-automatic mechanisms.

A number of recent efforts focused on semi-automatic schema integration or merging, including Chiticariu et al. [14], Melnik et al. [45], Pottinger and Bernstein [52,53], Saleem et al. [56]. However, proposed solutions are not generic enough. Instead of generating a comprehensive system and providing a complete solution, each work focuses on a specific subject. It is most of the time assumed that correspondences among source schemas are already given. Furthermore, it is not clear how to use the results of schema integration, for example for query processing over the integrated schema.

The most challenging part of schema integration is the identification of correspondences between elements of schemas from heterogeneous databases, which is referred to as schema matching. Matching problem has been tackled in different domains besides schema matching, such as matching media data [13]. Schema matching itself is identified as a fundamental process in variety of areas. Besides schema integration, schema matching is required for data warehousing, data integration, and peer-to-peer data management. Furthermore, schema matching is also correlated with ontology matching. Ontology matching has been an active research area especially with the increasing popularity of Semantic Web and peer-to-peer networks. For example, Haase et al. [31] uses ontology matching for peer selection. A large number of ontology matching systems have been proposed, as surveyed in Choi et al. [15], Euzenat and Shvaiko [22], Kalfoglou and Schorlemmer [36].

Although large numbers of efforts have focused on providing access to distributed, autonomous, and heterogeneous databases, including the PEER system [1,59], the Mediator EnvirOnment for Multiple Information Sources project (*MOMIS*) [10], the *InfoSleuth* project [9], the *SIMS* project [4], the Observer system [46], the Stanford-IBM Manager of Multiple Information Sources (TSIMMIS) system [26], and the COntext INterchange (COIN) project [28], they mostly lack the step of semi-automated schema matching and thus require a large amount of manual work. Schema matching has been considered as a separate research problem and its related challenges have been addressed by a number of research and development projects, including Aumueller et al. [5], Do and Rahm [18], Giunchiglia et al. [27], Li and Clifton [39], Li et al. [40], Madhavan et al. [41], Melnik et al. [44], Miller et al. [47], Wang et al. [64]. In addition to these projects, there are several other efforts that consider some specific features of schema matching. For example, Bernstein et al. [11], Embley et al. [21], Rahm et al. [54]) focused on matching large schemas or extensibility of the developed schema matching system. Furthermore, the main goal of Doan et al. [19], Gal [24,25], Nottelmann and Straccia [50] is to manage uncertainty in schema matching. Each of these efforts considers a different aspect of uncertainty. For instance, Nottelmann and Straccia [50] proposes a probabilistic framework, which combines the machine learning, information retrieval, and heuristic techniques for learning a set of mapping rules. Uncertainty is an important subject especially considering large amounts of semantics involved in schemas. However, there are still open issues related to this subject, such as the need for adequate user interfaces, as addressed in Magnani and Montesi [42]. Furthermore, since their main focus is on uncertainty, other requirements of schema matching are not considered.

It is not reasonable to think of a fully automatic system for schema matching due to the huge amount of semantics involved in designing schemas. Nevertheless, most of the previous schema matching systems have provided limited solutions. For example, only a narrow set of matching algorithms are used by these systems. Although user interaction needs to be considered as a fundamental subject, either a very primitive or no Graphical User Interface (GUI) is provided. Furthermore, the past research has typically taken into account the schema matching as a separate problem area and has not combined it with the needed schema integration.

In this respect, the Semi Automatic Schema Matching and INTegration (SASMINT) approach and system [60–62] described in this paper is designed and implemented to enable semi-automated matching of relational schemas by using a combination of metrics and algorithms from Natural Language Processing (NLP) and Graph Theory. Furthermore, after the schema matching is executed, SASMINT allows its user to modify and approve the results. Once results are validated or corrected, the system then automatically performs the schema integration, by utilizing the results of schema matching and applying a number of schema derivation rules. The result of schema integration also requires user validation. Since it is not possible to determine all kinds of schema conflicts automatically, SASMINT provides a sophisticated GUI for modifying both the match and the integration results. All user interaction with the system is handled by means of this GUI. Finally, results of schema matching and integration are captured and formulated for persistent storage as the SASMINT Derivation Markup Language (SDML) representation. Compared to previous work, addressed above, there are several main contributions of SASMINT, which can be summarized as follows:

- *Extending existing schema matching approaches and elevating the accuracy of schema matching*: SASMINT extends the existing approaches for schema matching such that it requires minimal user input and advances the level of results accuracy, by utilizing a large set of metrics and algorithms from NLP and Graph theory and combining them with a weighted sum.
- *Enabling semi-automatic identification of suited weights for algorithms*: It is important to assign an appropriate weight to each NLP metric and algorithm used in schema matching, bearing in mind that each metric is suitable for different types of strings. SASMINT provides the SAMPLER component to semi-automatically identify the appropriate weight for each NLP metric or algorithm used in the system.
- *Enabling semi-automatic schema integration*: SASMINT interrelates schema matching results directly with the schema integration. Heuristic rules are defined that run on the results of the schema matching and generate the derivation formalism for an integrated schema automatically. We assess this as a novel contribution, providing a strong competitive edge for the research on the SASMINT system.
- *Definition and incorporation of an XML-based language for enabling unambiguous interpretation of schema match/integration results*: Within the SASMINT system, we did devise an XML-based derivation language, which we call as SDML, for capturing and persisting schema match and integration results. The value of this language is addressed in Sect. 4.1.
- *Enabling user-friendly interaction by means of a GUI editor*: It is not possible to automatically extract all types of semantic and resolve all kinds of structural conflicts. Therefore, a suitable user-friendly GUI editor is provided for supporting both modifications of the results of schema matching and schema integration processes as well as their storage for further use.

The rest of this paper is organized as follows: Section 2 addresses different types of schema conflicts. Section 3 introduces the SASMINT system and addresses its main functionalities. Section 4 describes the schema integration step of SASMINT. Section 5 gives some information about the experimental evaluation of the SASMINT system and provides the results of its comparison with the COMA++ system. Finally, Sect. 6 summarizes the main conclusions of the paper and presents possible future improvements.

2 Schema conflicts

Different types of conflicts or heterogeneities exist among databases, such as differences in schemas and data models. SASMINT focuses on the schema conflicts, categorized here as Linguistic and Structural. Linguistic (both syntactic and semantic) and Structural Conflicts frequently occur in schemas and they need to be handled during the schema integration process. Especially semantic and structural conflicts are difficult cases and thus challenging for automatic schema integration. Since fully automated solution is not possible for these cases, user intervention might be required for resolving these types of conflicts. Brief explanations about different types of Linguistic and Structural conflicts follow below:

(1) **Linguistic Conflicts**: Also called as Naming Conflicts, these types of conflicts arise from either syntactical or semantic discrepancies. Linguistic dictionaries can be used to identify and resolve these differences. Different types of linguistic conflicts are listed below:

- *Syntactic*: Syntactic conflicts correspond to differences in the formats of the names. Examples include using stop words and special characters in the names.
- *Semantic*: Semantic conflicts are related to differences in the meaning of concepts. Two main kinds of semantic conflicts are the homonyms and synonyms. Homonyms occur when the same name is used for different concepts. It is difficult to identify homonyms by an automatic integration system. Synonyms occur when different names are used to refer to the same concept. Linguistic dictionaries containing the synonymy relationships among concepts can help to identify the synonyms.

(2) **Structural Conflicts**: Structural conflicts arise as a result of using different modeling constructs or integrity constraints. For example, the same concept might be represented by different modeling constructs in different schemas (referred to as the type conflict), or two or more entities might be related with different dependencies in different schemas (referred to as the dependency conflict). Another example is using different keys for the same concept in different schemas (referred to as the key conflict).

3 The SASMINT system

The SASMINT approach and its implementation, the SASMINT system, are proposed to eliminate the limitations of the previous schema matching and integration systems. It enables semi-automatic schema integration by using the results of schema matching for integration. In addition to being used for integrating schemas for generating a federated or global schema, SASMINT can also be used for generating mappings between a common schema and local schemas of participating databases, making it applicable for different application cases in different domains.

SASMINT achieves schema integration in two main steps: (1) Schema Matching: correspondences between two schemas are identified by resolving syntactic, semantic, and structural conflicts automatically and then the user is asked for the final validation, (2) Schema Integration: schemas are integrated or merged by using the results of schema matching and exploiting a number of integration rules. User validation is required on the results of schema integration also.

Main flow of information in the SASMINT system is shown in Fig. 1. There are four main activities in this flow: (1) Assigning weights to the metrics and algorithms and identifying the

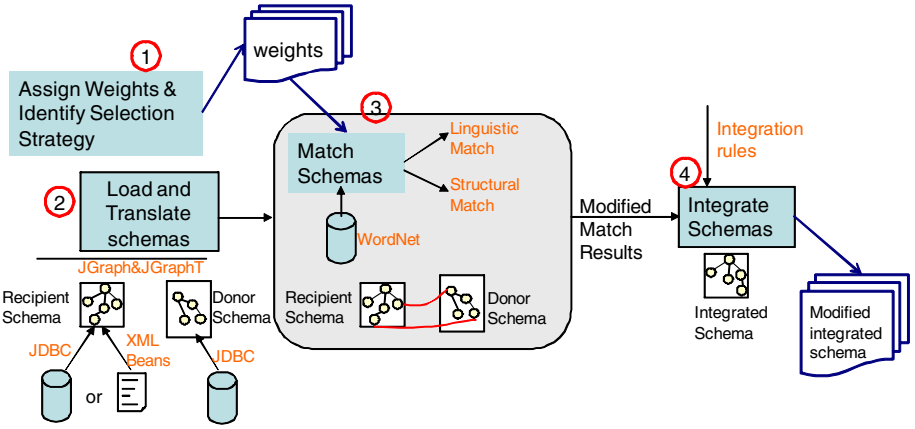


Fig. 1 Information flow in SASMINT

selection criteria, (2) Loading and translating schemas, (3) Matching Schemas, (4) Integrating Schemas. Details of these activities are explained below.

3.1 Assigning weights and identifying the selection criteria

This activity corresponds to the *Configuration step* and has two main goals:

(1) *Assignment of weights to all metrics and algorithms used in the schema matching*: As an improvement over previous schema matching approaches, which typically employ a limited number of algorithms, both Linguistic and Structure Matching components of SASMINT utilize a combination of different metrics and algorithms suitable for broad ranges of schemas to find out good match candidates. For this purpose, a weighted sum of these metrics and algorithms is calculated. By default, SASMINT assumes an equal weight distribution. Alternatively, user can manually assign weights through the user interfaces provided by SASMINT. Considering that it might be difficult to identify an appropriate weight for each metric manually, the *Sampler* component of SASMINT can be used in order to automatically calculate the appropriate weights for the Linguistic Matching metrics. In order for doing this, the user is required to provide, through the GUI of Sampler, up to five syntactically or semantically similar pairs depending on whether he wants to calculate the weights for syntactic or semantic similarity metrics. Then, Sampler uses syntactic or semantic similarity metrics to determine the similarity values that each metric produces. After finding out the similarity values, Sampler computes the F-measure value for each metric to assess the quality of the metric. Details about the F-measure are given in Sect. 5. This F-measure value is used by Sampler in the following formula, in order to determine the weight of each metric [61]:

$$w_m = \frac{1}{\sum F} * F_m$$

where $\sum F$ represents the sum of F-measure values resulted for all metrics used, and F_m represents the F-measure value calculated for metric 'm'. Higher F-measure value for a metric (thus higher quality) means higher weight for that metric.

(2) *Identifying the strategy for selecting the match results*: In order to achieve this goal of the configuration step, user input is required. User needs to provide a threshold value and choose one of the strategies for selecting the match results: (a) selecting all matching pairs

with similarity above the threshold (called as "select all above threshold") and (b) selecting the ones with the highest similarity value, if there is more than one element matching another element (called as "select max above threshold").

3.2 Loading and translating schemas

After assigning weights to the metrics and algorithms and identifying the selection strategy, two schemas, called the *recipient* and *donor* respectively, are loaded by the user into SASMINT by means of its GUI. Recipient schema is taken as the base schema in the integration process. It can be loaded either from a database or from an XML file, which contains a previously generated integrated schema and the related derivation information. Format of this file is described in the following paragraphs about the SDML. It is assumed that XML file contains an SDML-based representation of an integrated schema, which is always loaded as the recipient schema. On the other hand, donor schema can only be loaded from a database. During the loading process, schemas described in the language of their database are translated into a Directed Acyclic Graph (DAG) format. In other words, table and column names as well as primary and foreign keys are represented in graphs for the recipient and donor schemas. When a recipient schema is loaded from an XML file, only this information is shown in the graph, not the derivation information. DAG is the common format for representing schemas. SASMINT uses JGraphT, a free Java graph library [34], to create the DAG. When the two graphs, corresponding to donor and recipient schemas are generated, they are displayed by means of the SASMINT GUI. A graph component, called JGraph and its subcomponent, JGraph Layout are used for graph visualization and layout [35]. This activity of SASMINT, responsible for loading schemas and translating them into the graph format is called as *Preparation*.

3.3 Matching schemas

After loading and translating the recipient and donor schemas, user performs the 'Match' operation to identify the correspondences between these schemas. Schema Matching in SASMINT considers syntactic, semantic, and structural differences among schemas, in order to automatically identify likely matches between them. Schema matching starts with the pre-processing activity. Pre-processing brings schemas into a common form, by applying the following operations: (1) *Elimination of stop words and special characters*: Stop words, like "of", "the", and "an" and special characters like "_" are replaced with a space. (2) *Tokenization and Word Separation*: Strings containing more than one word are split into words. A change in the case or a space signifies a new token. (3) *Abbreviation Expansion*: Abbreviations are expanded. For this purpose, a list of well-known abbreviations is used and whenever a word is found in this list, it is replaced with its long form. (4) *Normalizing terms to a standard form using Lemmatization*: By means of lemmatization, verb forms are reduced to the infinitive and plural nouns are converted to their singular forms. WordNet, details of which are given in the following paragraphs, is used for finding a lemma of a word. For example, lemma of the word "scarves" is "scarf".

After the pre-processing activity, which enables to resolve several syntactic conflicts, two schemas are compared both *Linguistically* and *Structurally*. The result of schema matching for a pair of elements (a, b) is calculated by the following formula:

$$sim(a,b) = w_{\text{linguistic}} * sm_{\text{linguistic}}(a,b) \; + \; w_{\text{structural}} * sm_{\text{structural}}(a,b)$$

```
Inputs: S1in Graph Format, S2 in Graph Format
List_of_Nodes_S1 = getAllNodeNames (S1)
List_of_Nodes_S2 = getAllNodeNames (S2)
for each pair P(n1,n2) in List_of_Nodes_S1X List_of_Nodes_S2
    preprocessed P'(n1,n2) = preprocess (P(n1,n2))
    syn = SyntacticMatch(P'(n1,n2))
    if (syn < threshold)
       sem = SemanticMatch(P'(n1,n2))
    endIf
    else
       LinguisticMatch = syn
       break
    endElse
    if (sem > threshold)
       LinguisticMatch = sem
    endIf
    else
       LinguisticMatch = weight(syntactic) * syn+weight(semantic)*sem
    endElse
endFor
```

Fig. 2 Pseudo code of linguistic matching

where $sm_{\text{linguistic}}(a, b)$ is the linguistic similarity of (a, b), $w_{\text{inguistic}}$ is the weight of linguistic matching, $sm_{\text{structural}}(a, b)$ is the structural similarity of (a, b), and $w_{\text{structural}}$ is the weight of structural matching.

3.3.1 Linguistic matching

Linguistic Matching in SASMINT considers only the names of schema elements and compares the element name pairs from two schemas to resolve their syntactic and semantic conflicts and calculate their similarities by using the metrics and algorithms from the NLP domain. A pseudo code of linguistic matching is given in Fig. 2. Syntactic and semantic similarity metrics used by SASMINT are listed below:

(a) **Syntactic similarity**: Among different metrics from NLP available for comparing two strings syntactically, we have selected for SASMINT a number of well-known ones that are suitable for different types of strings. Such metrics are also used in the Information Retrieval (IR) domain for determining the document similarities, as addressed in Aygün [6], Wan [63]. Combining the results of syntactic similarity metrics using a weighted summation makes SASMINT applicable for different domains, which typically consist of schema elements in varying forms. This feature is necessary for achieving more accurate results. The metrics utilized by SASMINT are as follow:

- Levenshtein Distance [38]: It is based on the idea of minimum number of modifications required to change one string into another. This metric is string-based, which means that it considers strings as single strings even if they contain more than one word.
- Monge and Elkan [48]: It is a string-based distance function using an affine gap model. Monge–Elkan Distance allows for gaps of unmatched characters.
- Jaro [33]: It is a string-based metric intended for short strings and considers insertions, deletions, and transpositions. It also takes into account typical spelling deviations.
- Term Frequency*Inverse Document Frequency (TF*IDF) [57]: It is a vector-based approach from the information retrieval literature that assigns weights to terms. For each

of the document to be compared, first a weighted term vector is composed. Then, the similarity between the documents is computed as the cosine between their weighted term vectors.
- Jaccard Similarity [32]: It is a token-based similarity metric, meaning that it splits strings containing more than one word into tokens. It considers the number of shared words between two strings being compared.
- Longest Common Substring (LCS): It calculates the longest run of characters that appear in order inside both strings being compared.

(b) **Semantic similarity**: For comparing two strings semantically, different types of *path-based*, *information content,* and *gloss-based* measures from the NLP domain were examined in our study. However, we selected for SASMINT only path-based and gloss-based measures, as information content measures require a corpus and the most well-known corpuses are not available for free. Moreover, the choice of information content source can have a high impact on the results that one gets from the information content-based measures, making the selection of the most suitable corpus difficult.

Both path-based and gloss-based measures utilize WordNet. WordNet is a lexical dictionary, partitioned into nouns, verbs, adjectives, and adverbs [23]. These are organized into synonym sets. Synonym sets, also called as synsets, are interlinked by different relations, such as hypernymy, hyponymy, antonymy, meronymy, and holonymy. A brief description of each sense of a word, called as gloss, is also provided in WordNet.

As an example, consider the word "building". WordNet contains four senses for this word with the following gloss information: "a structure that has a roof and walls and stands more or less permanently in one place", "the act of constructing or building something", "the commercial activity involved in constructing buildings", and "the occupants of a building".

SASMINT's semantic similarity measures benefit from the IS-A hierarchy and gloss information provided by WordNet. Among the alternatives for the path-based and gloss-based measures, we have chosen two widely known measures that are explained below:

- *Path-based measure:* SASMINT utilizes the measure of Wu and Palmer [65]. Like other path-based measures, this measure calculates the shortest path between two concepts being compared, in a hierarchy. Since in our application we deal with element names, which are mainly nouns, and hypernymy/hyponymy (IS-A) is the dominant relationship linking nouns, we only consider a hierarchy containing this relationship when identifying semantic similarity of element names. In SASMINT, WordNet's IS-A hierarchy is used to identify the path between two concepts. For example, if "employee" and "person" are compared using Wu and Palmer's measure, their semantic similarity is found as 0.75, when WordNet 2.0 is used.
- *Gloss-based measure:* The algorithm of Lesk [37] is used as the base for the gloss-based measure. The gloss-based measure calculates the overlaps between the glosses of element names. SASMINT uses the gloss information provided by WordNet. For example, when "employee" and "person" are compared using the gloss-based measure, their similarity is calculated as 0.14 in WordNet version 2.0.

3.3.2 Structural matching

After schema elements are compared linguistically, SASMINT considers the structural aspects of schemas in *Structural Matching*, which uses the result of linguistic matching. It is based on the idea that if two elements have been found to be similar, their adjacent elements (parent and child nodes) may also match. Structural matching benefits from a variety of

graph similarity and matching algorithms from Graph Theory and Web searching, in order to resolve structural conflicts. Graph similarity algorithm of Blondel et al. [12] and the structure matching algorithm of Similarity Flooding [44] are selected for structural matching in SASMINT. The result of structural matching for a pair is the weighted sum of the results of these two algorithms.

3.3.3 Result generation after schema matching

After schema matching in SASMINT has identified similarities between each pair of schema elements from two schemas, results are displayed to the user. The user is required to make any required modifications on the results, because not all the mappings can be identified automatically by the system. After modifying and saving the results of matching, the user continues with the *Schema Integration*. Schema integration utilizes the results of schema matching and applies a number of heuristic rules to determine the integrated schema. The proposed integrated schema also requires the user validation. Details of Schema Integration in SASMINT are provided in the next section.

4 Schema integration with SASMINT

Schema integration is needed for generating both the federated schema in fully federated databases and also an integrated schema from the schemas of local nodes. Two types of strategies are mentioned in Batini et al. [8] for schema integration: *binary* and *n-ary* strategies. Binary strategies allow the integration of two schemas at a time, while n-ary strategies can integrate *n* schemas at a time. Because of the complexities of integrating *n* schemas at a time, most of the approaches in the literature prefer a binary strategy.

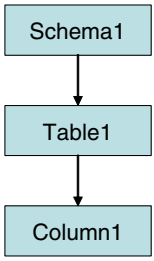
SASMINT also follows a binary strategy for schema integration and facilitates integration by providing some automatic means. It utilizes the results of schema matching. For every two schemas, after saving the results of their validated matching, user has the option to generate an integrated schema. Schema integration is a difficult process because of the structural and linguistic conflicts among schemas. Performing schema integration as automatically as possible is a significant improvement over the existing schema integration approaches.

4.1 SASMINT derivation language

Several derivation constructs are defined in SASMINT for representing the integrated schemas. These constructs are variations of the ones used by the PEER derivation language [2].

Two main types of derivation constructs are defined in SASMINT for relational schemas: (1) Table Derivation: consists of the derivations of type “Table Rename”, “Table Union”, “Table Subtract”, and “Table Restrict”, (2) Column Derivation: comprise the derivations of type “Column Rename”, “Column Union”, and “Column Extraction”. Table Rename, Table Union, Column Rename, Column Union, and Column Extraction are the ones typically used by automatic schema integration component of SASMINT.

Based on the derivation constructs, an XML-based derivation language is generated for capturing and storing the results of both schema matching and schema integration. This language is called as SDML. SDML has a format similar to other existing XML-based formats, such as Graph eXchange Language (GXL) [30] and GraphML [29], but is extended to store the results of both matching and integration. XML is chosen in SASMINT for representing




```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:sgraph xmlns:ns1="http://namespaces.sasmint.org/2007/04/GraphModel">
   <ns1:snode ns1:id="urn:sasmint:schema:Schema1" ns1:name="Schema1"
ns1:type="SCHEMA">
   </ns1:snode>
   <ns1:snode ns1:id="urn:sasmint:table:Schema1:Table1" ns1:name="Table1"
            ns1:schema="Schema1" ns1:type="TABLE">
   </ns1:snode>
   <ns1:snode ns1:id="urn:sasmint:column:Schema1:Table1:Column1"
            ns1:name="Column1"    ns1:schema="Schema1" ns1:table="Table1"
            ns1:type="COLUMN">
   </ns1:snode>
   <ns1:sedge ns1:id="urn:sasmint:hastable:c55a2772-e985-4ca5-8a7a-2dcaa2a6c72c"
            ns1:sourceNodeId="urn:sasmint:schema:Schema1"
            ns1:targetNodeId="urn:sasmint:table:Schema1:Table1"
            ns1:type="HASTABLE"/>
   <ns1:sedge ns1:id="urn:sasmint:hastable:d3ac69d7-0caf-4d4c-89cd-b361d4fef635"
            ns1:sourceNodeId="urn:sasmint:table:Schema1:Table1"
            ns1:targetNodeId="urn:sasmint:table:Schema1:Table1:Column1"
            ns1:type="HASCOLUMN"/>
   </ns1:sgraph>
```

Fig. 3 Graph and its SDML representation of a simple schema

the integrated schema, as XML provides a flexible format for storing and exchanging graphs. Furthermore, there is a wide range of tools for parsing and querying XML.

The value proposition of this particular contribution, SDML, is multi-faceted. First, the persisted schema match/integration results enable the external systems/agents to unambiguously interpret/understand these results (these external systems/agents could consume this information for implementing federated query processing, etc.). Second, this generic format is understandable by the match/integration human agent in-the-loop. What this means is that this human agent can easily modify the results. Finally, the structure of the derivation language is designed to keep the derivation history (i.e. for an entity, the whole derivation tree is kept). This feature enables incremental schema integration.

In Fig. 3, the graph and SDML representations of a simple schema are shown. The root element of the SDML document is the *sgraph* element, which consists of two main sub-elements: *snode* and *sedge*, as explained below:

- **snode**: represents a node in the graph and may contain the derivation constructs as sub-elements. Examples for these derivation constructs are given in the coming paragraphs. The *snode* element consists of the following attributes:

  id is a unique value in the entire document.
  name represents the name of the node, which is the same as the name of the schema, table, or column that this node represents.
  type indicates whether the element that this node represents is of type schema, table, or column.
  schema represents the name of the schema, which the element that this node represents belongs to.
  table represents the name of the table, which the element that this node represents belongs to. This attribute is optional. If the node is of type *table* or *schema*, corresponding *snode* definition contains no *table* attribute.
  refTable exists if this node represents a foreign key column and contains as the value the id of the table that this node has a foreign key reference.

- **sedge** represents an edge of the graph and if this is an edge connecting two similar nodes, has a sub-element called *similarity*. The *similarity* sub-element contains the similarity value. The *sedge* element consists of the following attributes:

  id is a unique value in the entire document.
  sourceNodeId identifies the id of the source node of the *sedge* element.
  targetNodeId identifies the id of the target node of the *sedge* element.
  type indicates the type of the edge. The value of the *type* attribute is HASTABLE if the edge is from a schema node to a table node, HASCOLUMN if it is from a table node to a column node, and SIMILARTO if it is an edge representing the similarity of source and target.

Example given in Fig. 3 shows a graph and SDML representation of a simple schema. The class diagram corresponding to the complete features of SDML is given in Fig. 4. As shown in Fig. 4, *sgraph* contains one or more *snode* and zero or more *sedge* elements. Each *snode* consists of zero or one derivation construct, including *tableUnionDerivation*, *tableRenameDerivation*, *tableSubtractDervation*, *tableRestrictDerivation*, *columnRenameDerivation*, *columnUnionDerivation*, and *columnStringAdditionDerivation*.

An example for each type of derivation is provided below. Only the related part of the XML document is shown in the examples.

- *Table Rename Derivation* renames a table. It is used to specify that a table of the integrated schema is derived from a table of either donor or recipient schema by giving it a new name. An example is given below.

```
<graph:snode graph:id="urn:sasmint:table:INTEGRATED_1:student"
        graph:name="student" graph:type="TABLE" graph:schema="INTEGRATED_1">
    <graph:tableRenameDerivation>
        <graph:derivationNode graph:name="student"
                graph:id="urn:sasmint:table:targetsc:student" graph:schema="targetsc"/>
    </ graph:tableRenameDerivation >
  </graph:snode>
```

- *Table Union Derivation* is used to specify that a table in the integrated schema is the union of two or more tables in the recipient and donor schemas.

```
<graph:snode graph:id="urn:sasmint:table:INTEGRATED_1:person"
       graph:name="person" graph:type="TABLE" graph:schema="INTEGRATED_1">
    <graph:tableUnionDerivation>
        <graph:derivationNode graph:schema="sourcesc" graph:name="person"
              graph:id="urn:sasmint:table:sourcesc:person" graph:type="TABLE" />
        <graph:derivationNode graph:schema="targetsc" graph:name="contact"
              graph:id="urn:sasmint:table:targetsc:contact" graph:type="TABLE"/>
    </graph:tableUnionDerivation>
  </graph:snode>
```

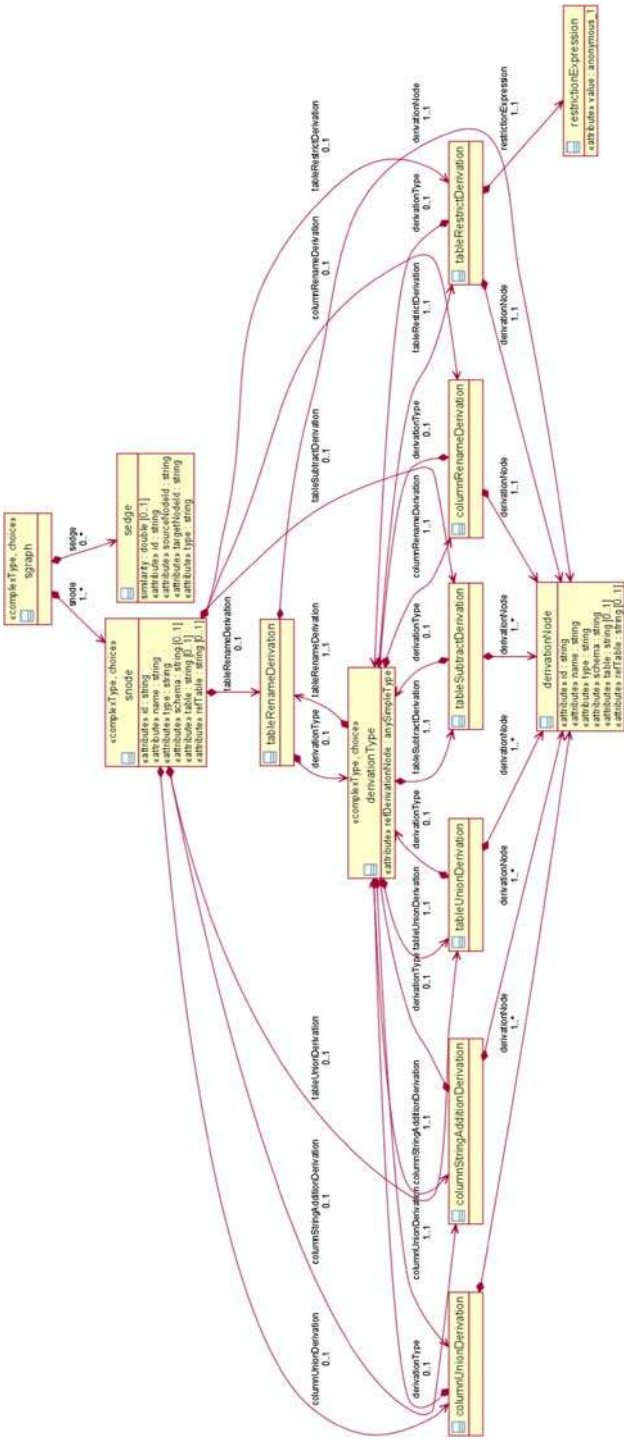


Fig. 4 UML class diagram of SDML

- *Table Subtract Derivation* is used to specify that a table in the integrated schema is constructed by subtracting a table in the recipient or donor schema from another table in the other schema.

```
<graph:snode graph:id="urn:sasmint:table:INTEGRATED_1:math_students"
        graph:name="math_students" graph:type="TABLE"
           graph:schema=" INTEGRATED_1">
    <graph:tableSubtractDerivation>
         <graph:derivationNode graph:schema="sourcesc " graph:name="students"
              graph:id="urn:sasmint:table:sourcesc:students" graph:type="TABLE" />
         <graph:derivationNode graph:schema="targetsc" graph:name="physics_students"
              graph:id="urn:sasmint:table:targetsc:physics_students "
                graph:type="TABLE"/>
    </graph:tableSubtractDerivation>
</graph:snode>
```

- *Table Restrict Derivation* is used to specify that a table in the integrated schema is derived from a table of either recipient or donor schema by applying a restriction.

```
<graph:snode graph:id="urn:sasmint:table:INTEGRATED_1:studentspassed"
      graph:name="studentspassed" graph:type="TABLE"
              graph:schema="INTEGRATED_1">
      <graph:tableRestrictDerivation>
              <graph:derivationNode graph:schema="targetsc" graph:type="TABLE"
             graph:id="urn:sasmint:table:targetsc:students" graph:name="students" />
                    <graph:restrictionExpression graph:value="grade>60"/>
              </graph:tableRestrictDerivation>
</graph:snode>
```

- *Column Rename Derivation* renames a column. It is used to specify that a column of the integrated schema is derived from a column of either donor or recipient schema by giving it a new name.

```
<graph:snode graph:id="urn:sasmint:column:INTEGRATED_1:person:contactid"
        graph:name="contactid" graph:type="COLUMN"
        graph:schema="INTEGRATED_1" graph:table="person">
     <graph:columnRenameDerivation>
        <graph:derivationNode graph:name="contactid" graph:table="contact"
            graph:id="urn:sasmint:column:targetsc:contact:contactid"
            graph:schema="targetsc"/>
     </graph:columnRenameDerivation>
 </graph:snode>
```

- *Column Union Derivation* is used to specify that a column in the integrated schema is the union of two or more columns in the recipient and donor schemas.

```
<graph:snode graph:id="urn:sasmint:column:INTEGRATED_1:person:phone"
        graph:name="phone" graph:type="COLUMN"
        graph:schema="INTEGRATED_1" graph:table="person">
     <graph:columnUnionDerivation>
        <graph:derivationNode graph:name="phone" graph:table="person"
            graph:id="urn:sasmint:column:sourcesc:person:phone"
            graph:schema="sourcesc"/>
        <graph:derivationNode graph:name="phoneno" graph:table="contact"
            graph:id="urn:sasmint:column:targetsc:contact:phoneno"
            graph:schema="targetsc"/>
     </graph:columnRenameDerivation>
 </graph:snode>
```

- Currently, one type of *Column Extraction Derivation* is defined, called as "*columnStringAdditionDerivation*". It is used to specify that a column in one schema equals to the concatenation of two or more columns in the other schema. An example for this case is shown below.

```
<graph:snode graph:id="urn:sasmint:column:INTEGRATED_1:person:name"
        graph:name="name" graph:type="COLUMN"
        graph:schema="INTEGRATED_1" graph:table="person">
    <graph:columnUnionDerivation>
      <graph:derivationNode graph:name="name" graph:table="person"
          graph:id="urn:sasmint:column:sourcesc:person:name"
          graph:schema="sourcesc"/>
      <graph:derivationNode graph:name="intname" graph:table="student"
          graph:id="urn:sasmint:column:targetsc:student:intname"
          graph:schema="targetsc"/>
       <graph:derivationType
           graph:refDerivationNode="urn:sasmint:column:targetsc:student:intname">
            <graph:columnStringAdditionDerivation>
              <graph:derivationNode graph:name="lname" graph:table="student"
              graph:id="urn:sasmint:column:targetsc:student:lname"
                      graph:schema="targetsc"/>
              <graph:derivationNode graph:name="fname" graph:table="student"
              graph:id="urn:sasmint:column:targetsc:student:fname"
                      graph:schema="targetsc"/>
            </ graph:columnStringAdditionDerivation>
        </graph:derivationType>
    </graph:columnUnionDerivation>
 </graph:snode>
```

4.2 Schema integration rules

Schema matching in SASMINT results in different types of matches as listed in Table 1. In order to automatically generate the integrated schema based on the results of schema matching, a number of heuristic rules have been defined for SASMINT. These rules cover the cases numbered as 1, 2, 3, 5, 7, 9, 10, 11, 12, 13, and 14 in Table 1. The remaining types of results correspond to highly complex cases and automatic solution is not possible for them.

As mentioned before, two schemas to be matched and integrated are called as recipient and donor, respectively. When deciding on which names to use for the column and tables of the integrated schema, the names in the recipient schema are considered first, but depending on the match case (for example, in the case of Column (n $\rightarrow$ 1) Column Y match), donor names may also be used. Integration process starts with table matches and continues with column matches. In other words, match pairs, one side of which is a table, are processed first. Tables that do not exist in any match pairs as well as their non-matching columns are directly added to the integrated schema. Therefore, all the tables in recipient and donor schemas are processed before their columns. In total, 13 rules are defined, as explained below. These rules are applied in this order. Some rules are similar, so details are provided here only for different cases, because of the space considerations.

Rule 1: This rule applies when a match is identified between one table (T_{r1}) of the recipient schema and m tables ($T_{d1...dm}$) of the donor schema. Its algorithm is represented as follows:

Begin

Generate a new table node, T_{i1}, based on the table T_{r1} of the recipient schema and add it to the integrated schema.

For each column of T_{r1} *and* each column of m tables, $T_{d1..dm}$, which do not match anything

If it is a foreign key column and the table that this column refers to is already covered in the derivation of a table T of the integrated schema, then add this column to the integrated schema as the column of the newly generated table T_{i1} *and add a reference to the table T.*

If it is not a foreign key column, then add the column to the integrated schema as the column of the newly generated table, T_{i1}

Apply the Table Union Derivation (***tableUnionDerivation***) operator to specify that the newly generated table, T_{i1} is the union of T_{r1} and $T_{d1..dm}$. Include this derivation in the integration result.

Apply the Column Rename Derivation (***columnRenameDerivation***) to the columns newly added to the integrated schema to specify that these columns of the integrated schema are the renamed versions of the related columns of T_{r1} and $T_{d1..dm}$.

End

Table 1 Match results

	Match result	Explanation	Covered
1	Column X (1 → 1) Column Y	Column X in the first schema matches Column Y in the second schema	Yes
2	Column X (1 → n) Column	Column X in the first schema matches n columns of the second schema	Yes
3	Column X (1→ 1) Table A	Column X in the first schema matches Table A in the second schema	Yes
4	Column X (1 → n) Table	Column X in the first schema matches n tables of the second schema	No
5	Column (m → 1) Column Y	n columns of the first schema match Column Y in the second schema	Yes
6	Column (m → n) Column	m columns of the first schema match n columns of the second schema	No
7	Column (m → 1) Table B	m columns of the first schema match Table B in the second schema	Yes
8	Column (m → n) Table	m columns of the first schema match n tables of the second schema	No
9	Table A (1→ 1) Table B	Table A in the first schema matches Table B in the second schema	Yes
10	Table A (1→ n) Table	Table A in the first schema matches n tables of the second schema	Yes
11	Table A (1→ 1) Column Y	Table A in the first schema matches Column Y in the second schema	Yes
12	Table A (1→ n) Column	Table A in the first schema matches n columns of the second schema	Yes
13	Table (m → 1) Table B	m tables of the first schema match Table B in the second schema	Yes
14	Table (m→ n) Table	m tables of the first schema match n tables of the second schema	Yes
15	Table (m→ 1) Column Y	m tables of the first schema match Column Y in the second schema	No
16	Table (m→ n) Column	m tables of the first schema match n columns of the second schema	No

Rule 2: This rule applies when a match is identified between one table (T_{d1}) of the donor schema and m tables ($T_{r1...rm}$) of the recipient schema. The algorithm for this rule is similar to the one for Rule 1, except that the new table node in the integrated schema is generated based on the table T_{d1} of the donor schema.

Rule 3: This rule applies when a match is identified between a table (T_{r1}) of the recipient schema and a table (T_{d1}) of the donor schema. Rule 3 is also similar to Rule 1, except that only one table of the donor schema is considered here as opposed to the m tables in Rule 1.
Rule 4: This rule applies when a match is identified between m tables ($T_{r1...rm}$) of the recipient schema and n tables ($T_{d1...dn}$) of the donor schema. Its algorithm is similar to Rule 1, except that instead of a single table of the recipient schema, m tables are considered here. For generating a new table node of the integrated schema, now we have m tables of the recipient schema, but this new table is again generated based on one of these tables (randomly selected asT_{r1}). Furthermore, when applying the **tableUnionDerivation** and **columnRenameDerivation**,m tables and their columns are taken into consideration, as opposed to the single recipient table in Rule 1.
Rule 5: This rule applies to the tables that are not involved in any match pair. All such tables and their columns that do not match anything are directly added to the integrated schema. Its algorithm is represented as follows:

Begin
Identify all non-matching tables, $T_{r1..rm}$ and $T_{d1..dm}$ in recipient and donor schemas respectively
For each $T_{r1..rm}$ and $T_{d1..dm}$
Generate a new table, T_{ix} and add it to the integrated schema
For each column of $T_{r1..rm}$ and $T_{d1..dm}$, which do not match anything
If it is a foreign key column and the table that this column refers to is already covered in the derivation of a table T of the integrated schema, then add this column to the integrated schema as the column of the newly generated table T_{ix} and add a reference to the table T.
If it is not a foreign key column, then add the column to the integrated schema as the column of the newly generated table, T_{ix}
Apply Table Rename Derivation (***tableRenameDerivation***) to specify that the newly generated tables are the renamed versions of $T_{r1..rm}$ and $T_{d1..dm}$.
Apply the Column Rename Derivation (***columnRenameDerivation***) to the columns newly added to the integrated schema, to specify that these columns of the integrated schema are the renamed versions of the related columns of tables $T_{r1..rm}$ and $T_{d1..dm}$ of the recipient and donor schemas.
End

Rule 6: This rule applies when a match is identified between a table (T_{r1}) of the recipient schema and m columns ($C_{d1...dm}$) of a table of the donor schema. Its algorithm is represented as follows:

Begin
Generate a new table node, T_{i1}, based on the table T_{r1} of the recipient schema and add it to the integrated schema.
For each column of the table T_{r1} of the recipient schema, which do not match anything
Add the column to the integrated schema as the column of the newly generated table, T_{i1}
For each column of table T_{r1} of the recipient schema, which is added to the integrated schema in the previous step
*Add Column Union Derivation (**columnUnionDerivation**) to specify that the newly generated column of the integrated schema is the union of the this column and m* columns ($C_{d1..dm}$) *of the donor schema*
Apply Table Rename Derivation (***tableRenameDerivation***) to specify that the newly generated table is the renamed version of table T_{r1}.
End

Rule 7: This rule applies when a match is identified between a table (T_{r1}) of the recipient schema and a column (C_{d1}) of a table of the donor schema. Its algorithm is similar to the one for Rule 6, except that only one column of the donor schema is processed in this rule as opposed to m columns in Rule 6.

Rule 8: This rule applies when a match is identified between a table (T_{d1}) of the donor schema and m columns ($C_{r1...rm}$) of a table of the recipient schema. The algorithm for this rule is very similar to the algorithm for Rule 6. The only difference is that in this rule, a table of the donor schema takes the place of the table of the recipient schema in Rule 6 and columns of a table of the donor schema take the place of the columns of a table in the recipient schema in Rule 6.

Rule 9: This rule applies when a match is identified between a table (T_{d1}) of the donor schema and a column (C_{r1}) of a table of the recipient schema. Similar to the cases for Rules 6–8, the algorithm for Rule 9 is very much like the algorithm for Rule 7.

Rule 10: This rule applies to the columns of the tables that are not involved in any match pair. Although such columns are processed in Rule 5 also, this rule is required in case there are any columns of the tables that are unprocessed in Rule 5. Its algorithm is represented as follows:

Begin

For each column of $T_{r1..rm}$ and $T_{d1..dm}$ of recipient and donor schemas, which do not match anything and not processed before

Identify its original parent table in the integrated schema. Search all table derivations to find out where this table exists and identify the related table T_{ix} in the integrated schema

If it is a foreign key column and the table that this column refers to is already covered in the derivation of a table T of the integrated schema, then add this column to the integrated schema as the column of the table T_{ix} and add a reference to the table T.

If it is not a foreign key column, then add the column to the integrated schema as the column of table, T_{ix}

End

Rule 11: This rule applies when a match is identified between a column (C_{r1}) of a table (T_{r1}) in the recipient schema and m columns ($C_{d1...dm}$) of a table (T_{d1}) of the donor schema. Its algorithm is represented as follows:

Begin

Identify the parent table T_{r1} of the column C_{r1} in the integrated schema. Search all table derivations to find out where this table T_{r1} exists and identify the related table T_{i1} in the integrated schema.

Generate a new column C_{i1} based on the C_{r1} and add it to the integrated schema as the column of T_{i1}.

Check whether C_{r1} is a foreign key column. If so, search all table derivations to find out the table that this column refers to and identify the related table T in the integrated schema. Add a reference to this table T from the newly generated column, C_{i1}.

Check whether a derivation rule is specified by the user after schema matching, for these m columns $C_{d1..dm}$ of the donor schema.

If no rule is specified, apply Column Union Derivation (***columnUnionDerivation***) to specify that the newly generated column C_{i1} is the union of C_{r1} and $C_{d1..dm}$

If Column String Addition Derivation (***columnStringAdditionDerivation***) is defined, apply this integration rule to the columns $C_{d1..dm}$ to get an intermediary column C_{x1}. Then, apply Column Union Derivation to specify that the newly generated column C_{i1} is the union of C_{r1} and C_{x1}

End

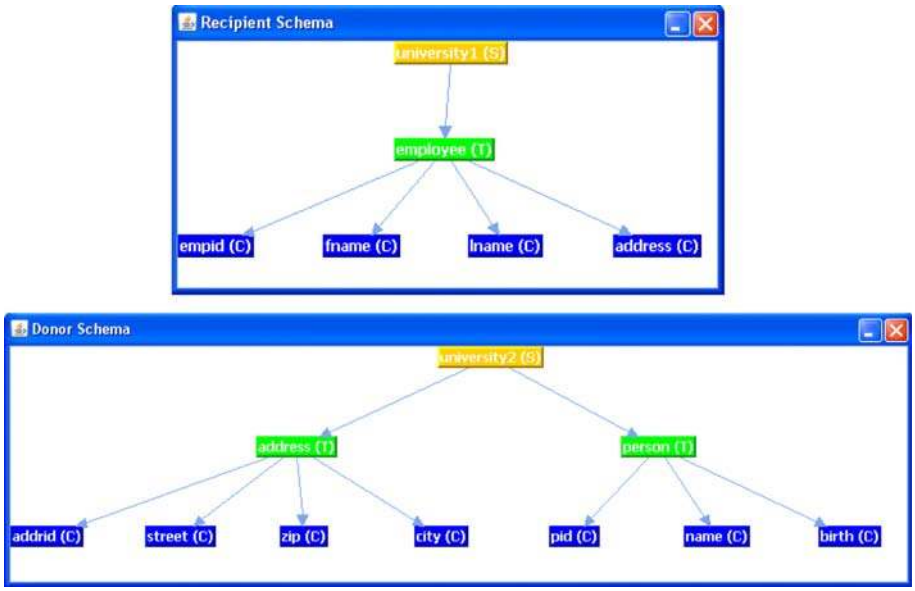


Fig. 5 Recipient and donor schemas in graph format

Rule 12: This rule applies when a match is identified between a column (C_{d1}) of a table (T_{d1}) in the donor schema and m columns ($C_{r1...rm}$) of a table (T_{r1}) of the recipient schema. Its algorithm is similar to the one for Rule 11, except that in this rule a column in the donor schema takes the place of the column of the recipient schema in Rule 11 and m columns of the recipient schema take the place of the m columns of the donor schema in Rule 11.

Rule 13: This rule applies when a match is identified between a column (C_{r1}) of a table (T_{r1}) in the recipient schema and a column C_{d1} of a table (T_{d1}) of the donor schema. Its algorithm is similar to the algorithm of Rule 11. The only difference that since in Rule 13 there is only one column that matches C_{r1}, there is no need to define a derivation like Column String Addition Derivation.

Derivations of type *tableUnionDerivation*, *tableRenameDerivation*, *columnRenameDerivation*, *columnUnionDerivation*, and *columnStringAdditionDerivation*, shown in bold in the algorithms above, are used in the automatic schema integration. Since it is difficult to automatically decide on the need for using *tableSubtractDerivation* and *tableRestrictDerivation* in the resulting integrated schema, these derivation types are not used in the algorithms. These derivation types can be used by the user when modifying the resulting integrated schema generated at the end of automatic schema integration.

After applying the rules, described above, an integrated schema is generated and the result is shown in both graph and XML format. Since two schemas are integrated at a time, XML file is expanded after each integration process with definitions of new nodes, edges, and derivations. As an example, consider that the recipient and donor schemas, shown in Fig. 5, are loaded into the SASMINT system. Assuming that the user validated schema matching is the one shown in Fig. 6; SASMINT automatically produces the integrated schema, shown in Fig. 7.

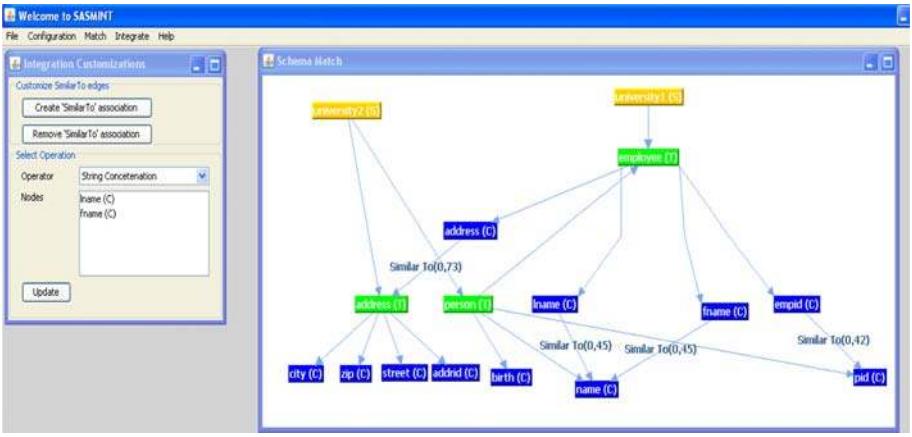


Fig. 6 Result of schema matching after user validation

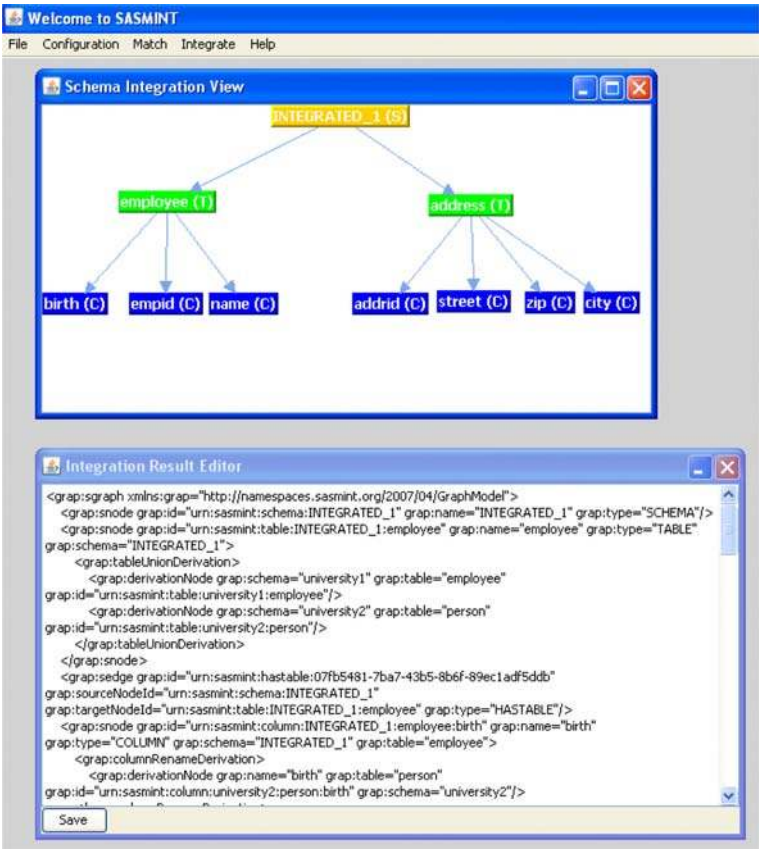


Fig. 7 Result of schema integration

5 Experimental evaluation of SASMINT

In order to evaluate the schema matching and integration in the SASMINT system, we carried out a number of experiments. Our main concern for the SASMINT system is the effectiveness, which means how accurately the system can identify matching pairs and the integrated schema automatically. Therefore, we only considered the quality measures in the experiments, not the performance measures. Performance measures depend on the underlying environment and technologies used, and thus it is difficult to obtain objective evaluations. Furthermore, when performance is considered, it is not only related to how fast the system works but also how much time the user spends to correct the results. When the system produces more accurate results, the user needs to spend less time and the overall performance increases.

We used two types of quality measures in our experiments: (1) quality measures for schema matching, and (2) quality measures for schema integration. Details of these measures are provided below.

As other schema matching evaluations do, we used the concepts of precision and recall from the information retrieval field [16] for measuring the quality of schema matching. Precision (P) and Recall (R) are computed as follows:

$$P = \frac{x}{x+z} \text{ and } R = \frac{x}{x+y}$$

where x is the number of correctly identified similar strings (i.e. true positives), z is the number of strings found as similar, while actually they were not (i.e. false positives), and y is the number of similar strings, which the system missed to identify (i.e. false negatives).

Although precision and recall measures are widely used for variety of evaluation purposes, neither of them can accurately assess the match quality alone. Therefore, a measure combining precision and recall is needed. F-measure [55] is one such a measure, combining recall and precision as follows:

$$F = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

Another measure, called as Overall, is proposed by Melnik et al. [44]. It is different from F-measure in that overall takes into account the amount of work needed to add the relevant mappings that have not been discovered (false negatives) and to remove those which are incorrect but have been extracted by the matcher (false positives). Overall, also called as accuracy, is defined by the following formula:

$$O = R * \left(2 - \frac{1}{P}\right)$$

Quality measures used for the assessment of schema integration in SASMINT benefit from the ideas presented in Batini et al. [8]. These measures are called completeness and minimality, as explained below:

(1) Completeness: Merged or integrated schema must cover concepts of all participating schemas.
(2) Minimality: If the same concept is represented in more than one participating schemas, integrated schema must contain only single representation of this concept. In other words, redundancies must be eliminated.

5.1 Experimental setup

We carried out experimental evaluation of schema matching using six pairs of relational test schemas; from purchase order (PO), hotel (HOTEL), biology (SDB), and university domains (UNIV-1, UNIV-2, UNIV-3). Hotel and SDB schemas are the modified versions of the ones used for the evaluation tests of MAPONTO [3] and UNIV-3 is the one used in the tests of the Similarity Flooding project. For the purpose of evaluating schema integration, three pairs of schemas from the university domain (UNIV-1, UNIV-2, UNIV-3) were integrated.

We compared the schema matching component of SASMINT against one of the state of the art schema matching systems, called COMA++ [5]. We used the COMA++ 2005 binary version that was the most recent version available at the time of our evaluation tests. We selected COMA++, because it was the most complete schema matching tool available at the time, providing a library of variety of matching algorithms and a sophisticated GUI. SASMINT and COMA++ are comparable as they both support matching of relational schemas and provide similar functionalities.

Before starting the evaluation tasks, we inserted a number of abbreviations and their expanded forms into the abbreviation lists of both systems. One important difference between SASMINT and COMA++ is that SASMINT uses WordNet for semantic matching, whereas COMA++ requires the user to add all synonyms in schema domains manually. Since WordNet might not contain all semantic relationships among the concepts of schemas either, we did not make any addition to the COMA++'s synonyms list, in order to make a fair comparison.

Combination of different metrics or algorithms was done in two systems as follows:

- **SASMINT**: We selected the default strategy for combining the metrics or algorithms, which is the weighted sum of them with equal weights used for each metric or algorithm (i.e. averaging). Although assigning appropriate weights for each match task would give better results, we decided to use the default strategy in order to make a fair comparison with COMA++.
- **COMA++**: We used the default matching strategy of COMA++, which is called as COMA. The COMA matcher combines the *name*, *path*, *leaves*, *parents*, and *siblings* matchers, by averaging them. In their tests, this combination was the winner and that is why we selected it.

As for the selection of match results, we used two different approaches that we call as "select all above threshold" and "select max above threshold", as detailed below:

(1) **Select All above Threshold**: Selecting all match pairs that have the similarity above a certain threshold value. We set the threshold as 0.5 in the experiments.

(2) **Select Max above Threshold**: Selecting the pairs with the maximum similarity. In SASMINT, whenever there is more than one concept matching a concept of a schema, the one with the higher similarity is selected as the matching candidate. If the difference between the similarity values is smaller than 0.01, then all such matches are selected. For clarity, if an element X has been found as similar to Y and Z with similarity values of 0.6 and 0.7 respectively, then only (X, Z) pair is shown in the results as similar. For the column-column matches, if there is a third element W that has been also found as similar to X with the value of 0.7, but if the parent table of W is not similar to that of X, while the parent table of Z is similar to that of X, then (X, Z) pair is selected as the similar pair. However, if there is no parent table similarity, both (X, Z) and (X, W) are selected as similar pairs. COMA++'s default selection strategy for the COMA matcher works as follows: When there is more than one match to the same concept, the one with

the higher similarity is selected if the difference between the similarity values is more than 0.0080, if not, all of the highest similarity matches are selected.

Although we compared SASMINT with COMA++ for the purpose of schema matching, we could not carry out functionality comparison for schema integration. COMA++ provides a simple schema merging functionality, but it is limited and not comparable to SASMINT's schema integration. To the best of our knowledge, there is no other system supporting both schema matching and schema integration. Therefore, we evaluated the integration component of SASMINT alone. For this purpose, we used six schemas from the university domain (UNIV-1, UNIV-2, UNIV-3).

5.2 Experimental evaluation

We carried out two types of experiments with SASMINT and COMA++: one using the "select all above threshold" strategy and another one using the "select max above threshold" strategy.

We saw that for both systems, the results for the "select all above threshold" strategy were worse than the results for the "select max above threshold" strategy. However, "select all above threshold" strategy is important when there is a need to suggest multiple candidates for each schema element and leave it to the user to identify the correct match among the alternatives. Instead of proposing only one match candidate for each schema element, which could be incorrect, the system suggests all possible match candidates, which makes it easier for the user to determine the final match result. The results of our experiments are provided below.

5.2.1 Evaluation of schema matching: using the "select all above threshold" strategy

When the precision measure is considered, the results were low for both systems. Low precision was due to the fact that for element names containing similar tokens, although the whole names were different, the final similarity result was usually above the threshold. The systems interpreted all tokens equally, while some tokens had no or little effect in the meaning. For example, "customer_contact" and "custCity" were identified as similar because "customer" tag matched "cust" tag (using the abbreviation expansion, "cust" was expanded as "customer"), but these names were actually not similar. Precision of SASMINT was on the average 0.58, whereas that of COMA++ was 0.26. This result was because of the high number of false positives identified by COMA++. In other words, COMA++ identified a large number of irrelevant matches, which can be a bigger problem when schemas being compared are large.

As for the recall measure, the result for COMA++ was on the average 0.92, whereas for SASMINT it was 0.86. However, this happened at the expense of very low precision values for COMA++. In order to achieve just a bit higher recall values, COMA++ sacrificed precision, resulting in very low precision values. This is because of the fact that there is an inverse relationship between the precision and recall. SASMINT missed some correct matches mostly due to low semantic similarity values it computed for some similar pairs, such as (product, item) and (suite, room). Especially the gloss-based measure was not as successful as we expected. Since the last version of WordNet (3.0) was not available yet for the Windows operating system, we had to use the previous version (2.0) of WordNet. We think that when the new version becomes ready, semantic similarity values for both path-based and gloss-based measures will be more correct.

When F-measure is considered, the difference between SASMINT and COMA++ is clearer, as shown in Fig. 8. This is due to the fact that F-measure is a combination of precision

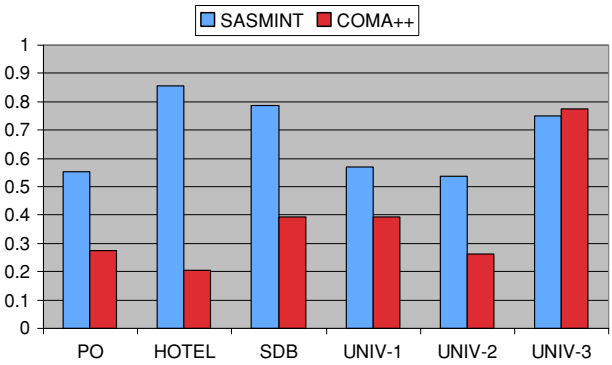


Fig. 8 Select all above threshold-results using F-measure

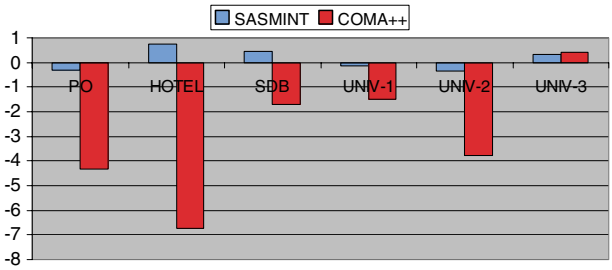


Fig. 9 Select all above threshold-results using Overall

and recall and although recall values for COMA++ were a bit higher than those for SASMINT, precision of SASMINT was much better than COMA++, which resulted in higher F-measure values for SASMINT.

On the average, overall values for neither SASMINT nor COMA++ turned out to be high, because of the wrongly identified and missed matches. However, since the number of false positive matches for COMA++ was very high, it had very low overall values, requiring too much manual intervention in order to remove these wrongly identified matches. Results based on the overall measure are shown in Fig. 9.

5.2.2 Evaluation of schema matching: using the "select max above threshold" strategy

Compared to the "select all above threshold" strategy, precision of "select max above threshold" strategy was very high for both systems. This was due to the fact that, in this second strategy only the most relevant matches were selected. On the average, SASMINT achieved the precision of 0.95, whereas the average precision of COMA++ was 0.93.

For SASMINT, recall values in the case of "select max above threshold" strategy were either the same or a bit lower than the ones in the "select all above threshold" strategy. For COMA++, recall was lower in the "select max above threshold" strategy for all schema pairs. On the average, SASMINT had the recall value of 0.76, whereas for COMA++ it was 0.72.

As for the F-measure, SASMINT and COMA++ accomplished almost the same for some test schemas, as shown in Fig. 10. However, for other schemas, SASMINT performed around

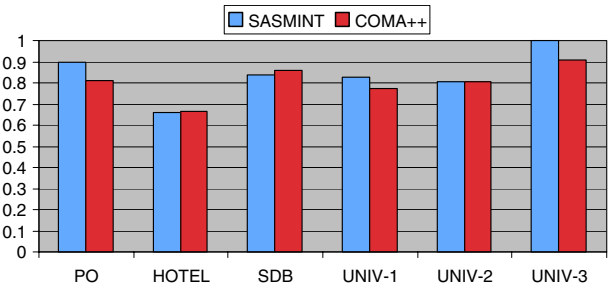


Fig. 10 Select max above threshold-results using F-measure

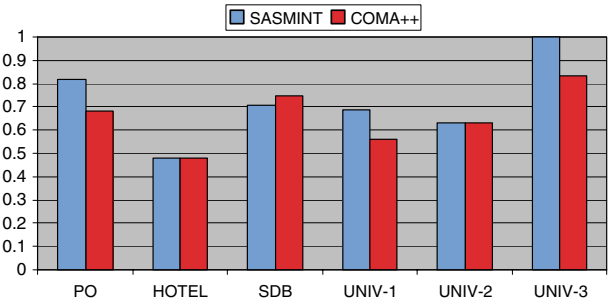


Fig. 11 Select max above threshold-results using Overall

1.1 times better than COMA++. The average F-measure for SASMINT was 0.84, whereas for COMA++, it was 0.80.

Situation for the overall measure was similar to F-measure. Average overall value for SASMINT was 0.72, whereas for COMA++, it was 0.65. Complete results for the overall measure are shown in Fig. 11.

5.2.3 Evaluation of schema integration

In order to evaluate the schema integration component of SASMINT, we used three schema pairs (i.e. 6 schemas) from the university domain. We had SASMINT integrate two university schemas at a time, incrementally generating the final integrated schema. Integration process used the given matches (i.e. correct matches) between schemas. After each integration step, we measured the completeness and minimality of the integrated schema generated. We saw that SASMINT produced integrated schemas that were complete and on the average around 0.99 minimal.

5.3 Summary of the results

A brief summary of the experimental evaluation is given below:

- Unlike SASMINT, which uses WordNet's IS-A hierarchy for identifying semantic relationships between schema elements, COMA++ requires a list of synonyms. Therefore, only synonymy relationship is considered and manual effort is required to update this list with pairs of synonyms from the domain of schemas, which makes COMA++ domain dependent.

- SASMINT and COMA++ both provide a library of matchers. SASMINT provides a Sampler component, which helps the user to identify the appropriate weight for each linguistic matching metric. On the other hand, COMA++ supports different alternatives for combining, aggregating, and selecting match results from different metrics, which need to be set manually. This feature makes it difficult for an inexperienced user to identify the best combination.
- Representation of schemas through the GUIs is different in two systems. COMA++ does not explicitly show foreign keys. Instead of showing the foreign key column, it displays the table being pointed by the foreign key. However, in some cases this functionality does not work as expected.
- SASMINT stores the results based on the SDML format. This allows the results to be used for decomposition of queries to be sent to local schemas as well as for semi-automatically generating an integrated schema of the recipient and donor schemas. COMA++ has an internal repository for the results, but the user can not identify in which format the results are stored and it is not clear how to use these results outside of the system.
- In order to compare the quality of schema matching in SASMINT and COMA++, we carried out experiments based on two types of result selection strategies: (1) select all above threshold and (2) select max above threshold. Two systems both performed better in the second approach. When the first approach was used, results for COMA++ were worse than those for SASMINT. For the second approach, the systems performed almost the same for some schema pairs, for the remaining pairs SASMINT was better than COMA++.
- Unlike schema matching, we could not compare the schema integration in SASMINT with any other system, as explained previously. Schema integration tests with SASMINT produced promising results. Generated integrated schemas were complete and on the average around 0.99 minimal.

To sum up, experimental evaluation for the schema matching and schema integration components of SASMINT produced promising results. During the experiments, we also verified that in addition to the quality of matching and integration, another key feature of SASMINT is the way that it persists the results. SASMINT's XML-based SDML format allows the results to be used for decomposition of queries to be sent to local schemas. Furthermore, since it is in a format easily readable and understandable by the user, results can be modified by the user without any difficulty.

6 Conclusion and future work

In this paper, we focus on an important challenge: semi-automatic schema matching and schema integration, for enabling data sharing among collaborating organizations. We address different types of linguistic and structural conflicts that need to be resolved when dealing with this challenge. Then, we introduce the SASMINT System to support schema matching and schema integration as automatically as possible. Unlike other approaches to database interoperability, which use a limited number of metrics and algorithms, SASMINT uses different types of metrics and algorithms for schema matching and thus generates more accurate results. It also provides the Sampler component for semi-automatically identifying the appropriate weight for each Linguistic Matching metric. The result of matching process is ultimately displayed to the user who can validate, modify, and store the final mappings. After that, SASMINT generates an integrated schema based on the validated match results

and using a number of derivation rules. As another contribution, an XML-based derivation language, called SDML, is defined to capture and persist the match and integration results.

In future work, we plan to support domain ontologies in addition to the use of WordNet. Therefore, if there is ontology available for that domain, SASMINT will use this ontology in order to identify semantic similarities, which will enable it to resolve more types of semantic conflicts. This ontology may also be automatically extended with newly identified semantic relationships. In this case, another important issue to consider is how to build ontologies [51]. As another future work, Sampler will be further extended to utilize the machine learning techniques. The "select all above threshold" strategy could be further extended to consider uncertainty and provide support for probabilistic schema matching, based on the active work in this field. Another future work could be creating a benchmark for schema matching and integration. Currently, there is no benchmark available, which necessitates each work to generate its own set of test schemas and do its own test for evaluation purposes. Therefore, creation of such a benchmark would be valuable.

References

1. Afsarmanesh H, Wiedijk M, Hertzberger LO et al (1996) Cooperation of CIM expert systems supported by PEER. J Stud Inf Control 5(2):157–169
2. Afsarmanesh H, Wiedijk M, Tuijnman F et al (1994) The PEER information management language user manual. Technical Report. Department of Computer Systems, University of Amsterdam
3. An Y, Mylopoulos J, Borgida A (2006) Building semantic mappings from databases to ontologies. In: Twenty-First National Conference on Artificial Intelligence (AAAI-06) Nectar Track, Boston
4. Arens Y, Knoblock CA, Shen W-M (1996) Query reformulation for dynamic information integration. J Intell Inf Syst 6(2/3):99–130
5. Aumueller D, Do HH, Massmann S et al (2005) Schema and ontology matching with COMA++. In: ACM SIGMOD international conference on management of data. ACM, Baltimore, pp 906–908
6. Aygün RS (2008) S2S: structural-to-syntactic matching similar documents. Knowl Inf Syst 16(3):303–329
7. Batini C, Lenzerini M (1984) A methodology for data schema integration in the entity relationship model. IEEE Trans Softw Eng 10(6):650–664
8. Batini C, Lenzerini M, Navathe S (1986) A comparative analysis of methodologies for database schema integration. ACM Comput Surv 18(4):323–364
9. Bayardo RJ, Bohrer W, Brice R et al (1997) InfoSleuth: agent-based semantic integration of information in open and dynamic environments. In: ACM SIGMOD international conference on management of data. ACM, Tucson, pp 195–206
10. Bergamaschi S, Castano S, Vimercati SDCD et al (1998) A semantic approach to information integration: the MOMIS project. In: Sesto Convegno della Associazione Italiana per l'Intelligenza Artificiale (AI*IA98), Padova, Italy
11. Bernstein PA, Melnik S, Petropoulos M et al (2004) Industrial-strength schema matching. SIGMOD Rec 33(4):38–43
12. Blondel VD, Gajardo A, Heymans M et al (2004) A measure of similarity between graph vertices: applications to synonym extraction and Web searching. SIAM Rev 46(4):647–666
13. Candan KS, Kim JW, Liu H et al (2006) Discovering mappings in hierarchical data from multiple sources using the inherent structure. Knowl Inf Syst 10(2):185–210
14. Chiticariu L, Kolaitis PG, Popa L (2008) Interactive generation of integrated schemas. In: ACM SIGMOD international conference on management of data. ACM, Vancouver, pp 833–846
15. Choi N, Song I-Y, Han H (2006) A survey on ontology mapping. SIGMOD Rec 35(3):34–41
16. Cleverdon CW, Keen EM (1966) Aslib–Cranfield research project. Technical Report. Cranfield Institute of Technology, Cranfield

17. Dayal U, Hwang H-Y (1982) View definition and generalization for database integration in multibase: a system for heterogeneous distributed databases. In: Berkeley workshop, pp 203–238
18. Do HH, Rahm E (2002) COMA—a system for flexible combination of schema matching approaches. In: International conference on very large databases (VLDB), VLDB Endowment. Hong Kong, China, pp 610–621
19. Doan AH, Domingos P, Halevy A (2001) Reconciling schemas of disparate data sources—a machine-learning approach. In: ACM SIGMOD international conference on management of data. ACM, Santa Barbara, pp 509–520
20. ElMasri R, Larson J, Navathe SB (1987) Integration algorithms for federated databases and logical database design. Technical Report. Honeywell Corporate Systems Development Division
21. Embley DW, Xu L, Ding Y (2004) Automatic direct and indirect schema mapping: experiences and lessons learned. SIGMOD Rec 33(4):14–19
22. Euzenat J, Shvaiko P (2007) Ontology matching. Springer, Heidelberg p 445
23. Fellbaum C (1998) An electronic lexical database. MIT press, Cambridge p 445
24. Gal A (2006) Managing uncertainty in schema matching with Top-K schema mappings. J Data Semant Special Issue Emerg Semant 6:90–114
25. Gal A (2007) Why is schema matching tough and what can we do about it. SIGMOD Rec 35(4):2–5
26. Garcia-Molina H, Papakonstantinou Y, Quass D et al (1997) The TSIMMIS approach to mediation: data models and languages. J Intell Inf Syst 8(2):117–132
27. Giunchiglia F, Yatskevich M, Shvaiko P (2007) Semantic matching: algorithms and implementation. J Data Semant 9:1–38
28. Goh C, Bresson S, Madnich S et al (1999) Context interchange: new features and formalisms for the intelligent integration of information. ACM Trans Inf Syst 17(3):270–293
29. GraphML (2008) http://graphml.graphdrawing.org/
30. GXL (2008) http://www.gupro.de/GXL/
31. Haase P, Siebes R, Harmelen Fv (2008) Expertise-based peer selection in peer-to-peer networks. Knowl Inf Syst 15(1):75–107
32. Jaccard P (1912) The distribution of flora in the alpine zone. New Phytol 11(2):37–50
33. Jaro MA (1995) Probabilistic linkage of large public health data files. Stat Med 14:491–498
34. JGraph (2008) http://www.jgraph.com/
35. JGraphT (2008) http://jgrapht.sourceforge.net/
36. Kalfoglou Y, Schorlemmer M (2003) Ontology mapping: the state of the art. Knowl Eng Rev J 18(1):1–31
37. Lesk M (1986) Automatic sense disambiguation using machine readable dictionaries: how to tell a pine code from an ice cream cone. In: 5th international conference on systems documentation. Toronto, Ontario, Canada, pp 24–26
38. Levenshtein VI (1966) Binary codes capable of correcting deletions, insertions, and reversals. Cybern Control Theor 10(8):707–710
39. Li W, Clifton C (2000a) SEMINT: a tool for identifying attribute correspondence in heterogeneous databases using neural networks. J Data Knowl Eng 33(1):49–84
40. Li W, Clifton C, Liu SY (2000b) Using neural networks: implementation and experiences. Knowl Inf Syst 2(1):73–96
41. Madhavan J, Bernstein PA, Rahm E (2001) Generic schema matching with cupid. In: International conference on very large databases (VLDB). Morgan Kaufmann, San Francisco, pp 49–58
42. Magnani M, Montesi D (2007) Uncertainty in data integration: current approaches and open problems. In: International VLDB workshop on management of uncertain data, pp 18–32
43. Mannino MV, Effelsberg W (1984) A methodology for global schema design. Technical Report, Computer and Information Sciences Department, University of Florida
44. Melnik S, Garcia-Molina H, Rahm E (2002) Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In: International conference on data engineering. IEEE Computer Society, San Jose, CA, USA, pp 117–128
45. Melnik S, Rahm E, Bernstein PA (2003) Rondo: a programming platform for generic model management. In: ACM SIGMOD international conference on management of data, pp 193–204
46. Mena E, Illarramendi A, Kashyap V et al (2000) OBSERVER: an approach for query processing in global information systems based on interoperation across pre-existing ontologies. Distrib Parallel Databases J 8(2):223–271
47. Miller RJ, Haas LM, Hernandez MA (2000) Schema mapping as query discovery. In: International conference on very large databases (VLDB). Morgan Kaufmann, Cairo, pp 77–88
48. Monge AE, Elkan C (1996) The field matching problem: algorithms and applications. In: Second international conference on knowledge discovery and data mining. AAAI Press, Portland, pp 267–270

49. Motro A, Buneman P (1981) Constructing superviews. In: ACM SIGMOD international conference on management of data, ACM, Ann Arbor, pp 56–64
50. Nottelmann H, Straccia U (2007) Information retrieval and machine learning for probabilistic schema matching. Inf Process Manage 43(3):552–576
51. Pinto HS, Martins JP (2004) Ontologies: how can they be built. Knowl Inf Syst 6(4):441–464
52. Pottinger R, Bernstein PA (2008) Schema merging and mapping creation for relational sources. In: International conference on extending database technology (EDBT). ACM, Nantes, pp 73–84
53. Pottinger RA, Bernstein PA (2003) Merging models based on given correspondences. In: International conference on very large databases (VLDB). Morgan Kaufmann, Berlin, pp 826–873
54. Rahm E, Do HH, Massmann S (2004) Matching large XML schemas. SIGMOD Rec 33(4):26–31
55. Rijsbergen CJV (1979) Information retrieval. Butterworth, London
56. Saleem K, Bellahsene Z, Hunt E (2008) PORSCHE: Performance ORiented SCHEma mediation. Inf Syst 33(7–8):637–657
57. Salton G, Yang CS (1973) On the specification of term values in automatic indexing. J Documentation 29:351–372
58. Sheth A, Larson J (1990) Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput Surv 22(3):183–236
59. Tuijnman F, Afsarmanesh H (1993) Management of shared data in federated cooperative PEER environment. Int J Intell Cooperation Inf Syst 2(4):451–473
60. Unal O, Afsarmanesh H (2006a) Interoperability in collaborative network of biodiversity organizations. In: 7th PRO-VE. Springer, Helsinki, pp 515–524
61. Unal O, Afsarmanesh H (2006b) SASMINT system for database interoperability in collaborative networks. In: OTM conferences, Lecture Notes in Computer Science. Springer, Montpellier, pp 91–108
62. Unal O, Afsarmanesh H (2006c) Using linguistic techniques for schema matching. In: International conference on software and data technologies. INSTICC Press, Setubal, pp 115–120
63. Wan X (2008) Beyond topical similarity: a structural similarity measure for retrieving highly similar documents. Knowl Inf Syst 15(1):55–73
64. Wang G, Goguen J, Nam Y et al (2004) Critical points for interactive schema matching. In: Sixth Asia Pacific web conference. Lecture Notes in Computer Science, Springer, pp 654–664
65. Wu Z, Palmer M (1994) Verb semantics and lexical selection. In: 32nd annual meeting of the association for computational linguistics. Association for Computational Linguistics, Las Cruces, pp 133–138

Author Biographies

Ozgul Unal received her Bachelors degree from the Department of Computer Engineering at Middle East Technical University in Turkey and a Masters degree from the Department of Information Systems at the same university. Since September 2002, she is a PhD student at the Computer Science Department of the Faculty of Science of the University of Amsterdam, in the Netherlands. She has been involved in several European and Dutch national research projects, focusing on the analysis, design and implementation of the Federated Information Management Systems in the domain of Bio-Sciences. Her current research areas include resolution of syntactic, semantic, and structural heterogeneities among database schemas in order to support (semi-) automatic schema matching and integration.

Dr. Hamideh Afsarmanesh is an associate professor of Computer Science at the University of Amsterdam in the Netherlands, where she heads the COLNET group at the Informatics Institute. She obtained her PhD from the University of Southern California (USC) in 1985. Her current research focus and projects address Federated Cooperative Databases and their automated schema integration, reference modeling and development of infrastructure/support tools for Virtual Organizations/Virtual Laboratories/Virtual Communities, and delivering proof of concepts applicable to domains such as Manufacturing, Tele-assistance, and Overdiversity. She is a member of the editorial board of the journals of *IJITM, IJASM, IJMLO, and IJEA*. She has been involved in initiation/organization of the PRO-VE and BASYS conferences. She has co-authored and co-edited more than 15 books, and published more than 150 articles. She is the chair of general assembly in Society of Collaborative Networks (SOCOLNET). She is the Dutch representative at the IFIP TC5, and the chair of WG5.5.