Schema Matching and Integration for Data Sharing Among Collaborating Organizations

Ozgul Unal
University of Amsterdam, The Netherlands
Email: O.Unal@uva.nl

Hamideh Afsarmanesh
University of Amsterdam, The Netherlands
Email: H.Afsarmanesh@uva.nl

***Abstract*-Schema matching and schema integration are important components of the data sharing infrastructure in Collaborative Networks. In order to achieve more accurate matching and integration results and enhance efficiency, it is required to provide some mechanisms to carry out these processes as automatically as possible. This paper addresses the problems and challenges related to schema matching and schema integration and introduces the Semi-Automatic Schema Matching and INTegration (SASMINT) system to automate these processes. Other systems aiming at database interoperability typically focus either on schema matching or on schema integration. On the other hand, the SASMINT system combines them and uses the results of schema matching for semi-automatic schema integration. SASMINT follows a composite approach in schema matching, which means it combines the results of variety of algorithms, making it a generic tool applicable for different types of schemas. It also proposes a Sampler component for helping the user to assign the weights to algorithms. Furthermore, SASMINT uses an XML-based derivation language to save the results of schema matching and schema integration, and also to define the components of integrated schemas, in order to further support automated query processing against integrated sources.**

***Index Terms*-Schema matching, schema integration, collaborative networks**

I. INTRODUCTION

With the advances of Internet, the number of information sources accessible through the Web is increasing. However, these advances create new challenges. For example, there is a huge amount of related data made available by distributed providers. Rather than accessing and manipulating single database systems in isolation, research is needed to make it possible to simultaneously access and manipulate different remote databases. In addition to being distributed, the voluminous data are exposed by various data providers (e.g. institutions, organizations, companies, etc.), which have their own proprietary data models resulting in heterogeneity among databases. In order to provide transparent access to such remote data and enable the sharing of information among heterogeneous and autonomous databases, their schema heterogeneity needs to be identified and resolved. Proposing a solution for such problems is more challenging for environments whose members shall collaborate, while they pose a number of heterogeneities that need to be addressed by the infrastructure. For example, when a number of organizations are members of collaborative networks, the proposed infrastructure must support them with sharing and exchange of their information.

More and more organizations understand the need to work together in order to better achieve their common goals. The importance of collaboration has been well understood in different domains, resulting in a rise in the number of collaborating organizations. A Collaborative Network (CN) is formed by variety of autonomous, geographically distributed, and heterogeneous organizations that collaborate to better achieve common or compatible goals [1]. Several forms of collaborative networks are evolving in parallel. Among the promising types of CNs, one can mention Virtual Organizations or Virtual Enterprises, Virtual Communities, and Virtual Laboratories.

It is important to provide an infrastructure enabling database interoperability, especially considering that collaborative networks need to be formed quickly [2]. Heterogeneity is the most important obstacle facing the collaboration. Since data sharing constitutes the main type of collaboration, the collaboration infrastructure has to consider such differences for providing effective mechanisms to integrate or inter-link and homogeneously access heterogeneous databases. However, automatic resolution of schema heterogeneity still remains a major bottleneck for provision of integrated data access/sharing among autonomous, heterogeneous, and distributed databases. In order to provide transparent access to such remote data and enable the sharing of information among databases, their schema heterogeneity needs to be identified and resolved and then the correspondences among schemas need to be identified. This process is called as schema matching. After schema matching, schemas might need to be also integrated, depending on the needs of the CNs. It is clear to see that schema

matching and schema integration constitute the key processes in information and communication technologies (ICT) infrastructures supporting collaboration. Tools that enable semi-automatic matching and integration are among the most important components of such infrastructures.

Both schema matching and schema integration are challenging, especially considering the naming and structural differences among schemas. In most previous approaches reported in literature, there is a great amount of manual work involved in schema matching and integration. Although there is some research focusing on semi-automatic schema matching (as later addressed in the related research section), it is not interlinked with the automation of schema integration. There is still need for clever and flexible user interfaces to display match results. Another limitation of the previous approaches is that they typically do not combine different match algorithms in a flexible way. Taking these limitations into account, we propose the SASMINT (Semi Automatic Schema Matching and INTegration) system and approach [3-5]. SASMINT proposes a solution to automate the processes related to interlinking of heterogeneous relational databases, particularly focused on schema matching and schema integration in collaborative environments including different forms of collaborative networks. Compared to other approaches in the literature, SASMINT combines a number of algorithms for semi-automatic schema matching and uses the result of matching for semi-automatic schema integration, needed for providing access to distributed, heterogeneous, and autonomous databases.

The rest of this paper is organized as follows: Section II introduces different types of information management systems aiming at providing access to distributed and heterogeneous databases. This section also summarizes different types of information related heterogeneities. Section III provides a background review of schema matching and schema integration. Section IV addresses the related work and open issues. Section V introduces the SASMINT system. Sections VI, VII, and VIII describe the Configuration, Schema Matching, and Schema Integration steps of SASMINT respectively. Section IX provides some discussions about the application of SASMINT through a small example. Finally, Section X summarizes the main conclusions of the paper.

II. Integrated Information Management and Heterogeneity

Enabling interoperability among distributed and heterogeneous databases has been a significant issue in different domains, including CNs. Different architectures have been proposed in the literature, concerning the management and sharing of data provided by distributed and possibly heterogeneous and autonomous databases. Many terms have been used to describe these architectures, such as multidatabase systems, federated and non-federated database systems, whereas there is no consensus of terminology in the database community. In order to provide our understanding of the terms, we provide a classification for such systems that we call as Integrated Information Management Systems, as shown in Fig. 1.

By following the definition of [6], we mention two types of integrated information management systems: distributed database systems and multidatabase systems. Based on the classification of [7], we divide the multidatabase systems as federated information management systems and non-federated information management systems.

Federated information management systems consist of nodes, which autonomously decide which part of their data to share with others. These systems can follow a fully federated schema or a global federated schema approach. As illustrated in Fig. 2, a fully federated schema approach constructs an integrated schema at each node by merging the local schema of that node with the schemas imported from other nodes. Import schemas represent the information that other nodes make available to this node. A global federated schema approach on the other hand, generates a global schema by integrating the export schemas (represent the shared part of information) from nodes into a single schema, as shown in Fig. 3.

Nodes of non-federated information management systems are not autonomous. Two approaches can be mentioned here: 1-to-1 schema mapping and common schema adaptation mapping. In 1-to-1 schema mapping approach, mappings between the schemas of nodes are

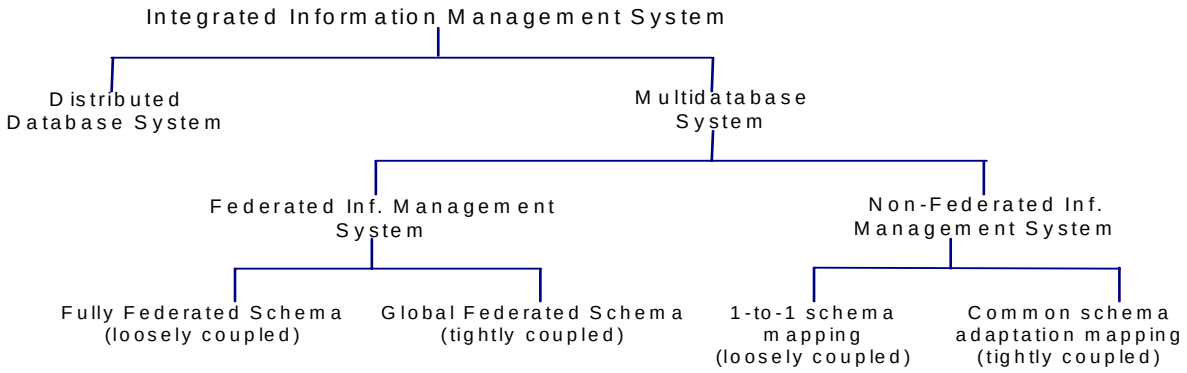

Figure1. Integrated Information Management System

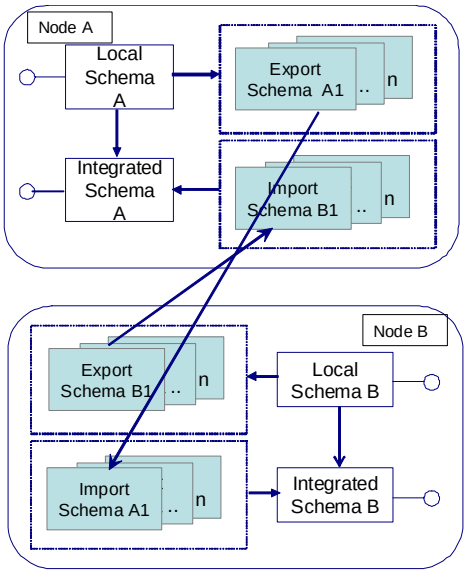

Figure 2. Fully Federated Schema

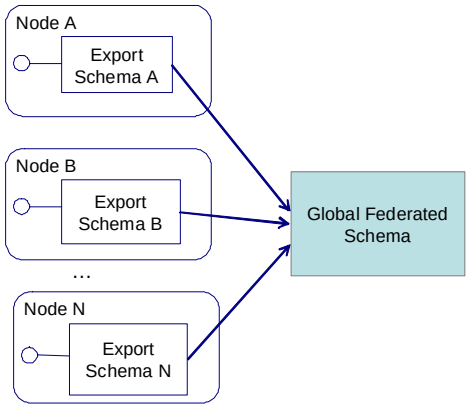

Figure 3. Global Federated Schema

identified in a pair-wise manner. For instance, as represented in Fig. 4, mappings between the schema of Node A and schemas of each other nodes are defined. Whereas in common schema adaptation mapping approach, mappings between the common schema and the local schema of each node are specified, as depicted in Fig. 5.

No matter which of the Integrated Information Management System approach is used in a network of collaborating organizations, heterogeneity is the main challenge to deal with. Heterogeneity exists at different levels, such as there might be differences in operating systems and in database management systems used, as well as in data definitions.

A number of classifications of heterogeneity have been proposed in the literature and there are many overlaps and discrepancies among these classifications. Considering the goals of SASMINT, introduced in this paper, we focus only on information related heterogeneities. Especially considering the differences in database schemas, we can mention the following types of heterogeneity:

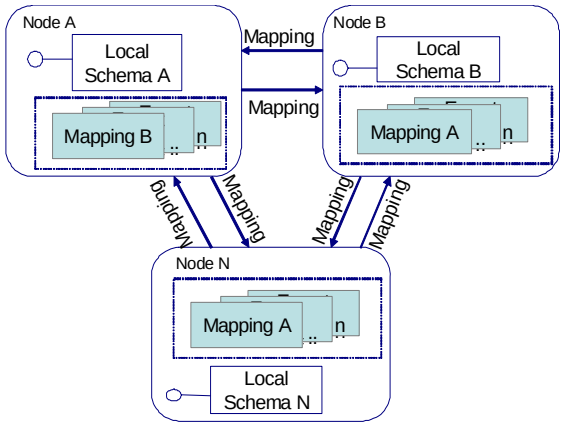

Figure 4. 1-to-1 Schema Mapping

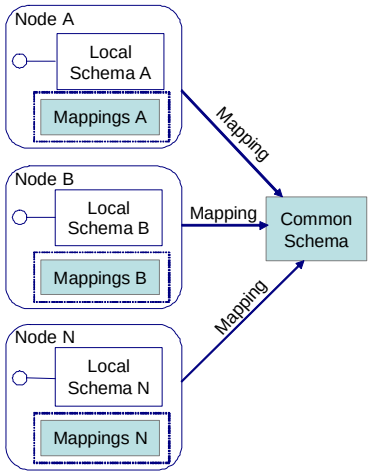

Figure 5. Common Schema Adaptation Mapping

1. **Structural Heterogeneity**: Different structural primitives are provided by different data models. For example, object-oriented data models support inheritance while relational data models do not (data model heterogeneity). Even if the data model is the same, similar information content may be represented differently in different schemas (schematic heterogeneity).

Following types are mentioned by [8] among the structural conflicts:

- **Type Conflicts:** These conflicts arise from using different modeling constructs (for example entity vs. attribute) for representing the same concept.
- **Dependency Conflicts:** These types of conflicts arise when concepts are related among themselves with different dependencies in different schemas, such as with 1-to-1 relationship in one schema, while 1-to-m relationship in another schema.
- **Key Conflicts:** This case arises when different keys are assigned to the same concept in different schemas.

2. **Syntactic Heterogeneity**: This type of heterogeneity is related to different formats used in the names of

the same concepts, such as using abbreviated vs. extended names.

3. **Semantic Heterogeneity**: This type is related to differences in meaning, dependent on the vocabulary and terminology used to express the information and the contexts in which it is interpreted.

There are two types of semantic relationships among the names used:

- Homonyms: The same name is used for two different concepts.
- Synonyms: The same concept is described by different names.

As it is clear from the existence of a large number of classifications, heterogeneity has been one of the fundamental problems in information systems. Among different types of heterogeneities mentioned in the literature, SASMINT system considers heterogeneities listed above. By combining the syntactic and semantic heterogeneity under the name *linguistic heterogeneity*, the research explained in this paper focuses on structural and linguistic schema conflicts. Especially structural conflicts are complex cases and cause difficulties for schema matching and integration algorithms. Since it is difficult to handle these cases automatically, user input is required.

III. Schema Matching and Schema Integration

Integrated information management systems, introduced in the previous section, need to tackle different types of heterogeneities in order to identify the correspondences among schemas, which is the aim of schema matching and integration. As a result, schema matching and schema integration have become two main processes in such systems.

Schema matching can be defined as finding correspondences between elements of two schemas. It plays an important role in several application domains, such as schema integration, data warehouses, query processing, Semantic Web, and e-business [9] [10]. The simplest type of matching is the 1-1 matching. For two schemas A and B, this type of matching identifies for each element of A the most similar element of schema B. In addition to 1-1 matches, complex matches also frequently occur among schemas. Complex matching finds out mappings between each element or a group of elements of schema A and a group of elements of schema B. Groups of elements are combined with a formula.

Schema Matching takes a variety of inputs and produces some outputs depending on the matching approach that it follows. Varieties of inputs consist of the schema information, a linguistic dictionary, a number of linguistic and structural similarity measures, and the user input. Output of matching is the similarity scores for each mapping identified.

The problem of schema integration in the context of distributed information systems is a relatively old problem. In different approaches for enabling access to distributed and heterogeneous data, different levels of integration might be required. In database research, schema integration is typically used to refer to both view integration and database integration [8]. View integration aims at producing an integrated schema of users' views and is performed during the database design process, whereas database integration derives a new schema from existing specification. As identified in [11], view integration methodologies work with views based on the same data model, but database integration technologies work with schemas that are usually defined using heterogeneous data models. Considering the goals of the research work explained in this thesis, database integration is the one being focused on and whenever schema integration is mentioned, database integration is meant.

Three steps are involved in schema integration: 1) Pre-integration, 2) Matching, and 3) Integration. The *Pre-integration* step consists of a number of preparations before the integration, such as identifying schemas to be integrated, preferences to be considered in the integration process, and amount of user input, as [8] mentioned. The *Matching* step, also called as the Investigation step by [11], identifies the correspondences among schemas by resolving the conflicts. The *Integration* step is responsible for integrating schemas based on the correspondences identified in the matching step.

IV. Related Work and Open Issues

Varieties of approaches for providing integrated data access/sharing among distributed, heterogeneous, and autonomous databases have been proposed in the literature. For example, the PEER is a generic object-oriented federated information management system enabling information sharing among autonomous and heterogeneous nodes [12]. There is an integrated schema for each node generated by integrating the local schema of the node and the schemas representing data that other nodes make available to this node. However, no automation is provided for generating this schema. In another project called SIMS (Services and Information Management for decision Systems) [13], in order to provide access to heterogeneous and distributed databases, first a common domain model is created using the Loom knowledge representation language. When an information source decides to join to the SIMS system, first its contents are modeled and then the concepts in information source model are related to the corresponding concepts of the domain model. Again, no automation is provided for this process. Similar to the PEER and SIMS systems, other efforts in this typically involve a large amount of manual work. They usually ignore the step of semi-automatic schema matching.

While interoperability has been an important topic in the database research, schema matching has been usually considered as a separate problem. A great deal of effort has been put into the study of increasing the degree of automation of schema matching. One such schema matching approach is proposed in the SEMINT (SEMantic INTegrator) system [14] that utilizes both schema and instance information. However, no Graphical User Interface (GUI) is provided. Cupid system [15] exploits a combination of name and structure matcher.

However, the name matcher uses only one string similarity metric and no GUI is provided. Similarity Flooding [16] converts diverse models into directed labeled graphs and then identifies the initial maps between elements of two graphs using only a simple string matcher. These initial maps are then used by a structure matcher. However, Similarity Flooding (SF) neither has the knowledge of edge and node semantics, nor it provides a GUI. Clio [17] generates alternative mappings as SQL view definitions based on the value correspondences that are defined by the user. No linguistic matching techniques are used and much manual work is required. S-Match [18] exploits a number of element and structure level match techniques. Result of schema matching is represented using the terms of equivalence, more general, less general, mismatch, and overlapping and no GUI is provided. COMA++ [19], which is a successor of COMA [20], provides a library of different types of matches and also a sophisticated GUI, making it more comprehensive than other systems. However, it is sometimes difficult for users to decide on the best combination of matchers.

As for schema integration, a number of systems or approaches have been introduced in the database literature. MOMIS (Mediator EnvirOnment for Multiple Information Sources) [21] has a component responsible for schema integration. However, it requires a database specialist to assists the integration process at each phase of integration. For example, it is necessary that all elements of schemas are annotated by the database designer manually using the appropriate meanings in the WordNet lexical database. COMA++, introduced above among the schema matching systems, provides functionality for schema merging, but since schema matching is the main focus of COMA++, schema merging is primitive and it is not possible to see how the elements of merged schema are derived from the local schemas and no mappings are defined between the merged schema and the local schemas. PORSCHE (Performance ORiented SCHEema mediation) [22] aims at creating a mediated schema from a set of large XML Schemas and identifying mappings from the source schemas to the mediated schema. It accepts a set of schema trees. PORSCHE has a linguistic matcher component, which uses tokenization, abbreviations, and synonyms. Abbreviation and synonym tables are generated by users. There is no GUI provided by PORSCHE and it is not clear how the results of integration are stored.

To summarize, although schema matching and schema integration have been the focus of large numbers of efforts in the literature, there are a number of issues, which are not sufficiently addressed yet and thus require further investigation:

- **Using a Combination of Match Algorithms:** Efforts in the schema matching research typically use a limited number of algorithms. However, in order to achieve high match accuracy, it is necessary to combine different types of algorithms, considering syntactic, semantic, as well as structural differences among schemas. Furthermore, in order to combine different algorithms, a weight needs to be identified for each of them. Identifying an appropriate weight for each algorithm is also an essential part of a system.
- **Graphical User Interface:** Developing algorithms for automatic schema matching is not sufficient alone. User interaction is another important topic to be considered when developing a schema matching and schema integration system. Especially considering that it is not possible to identify all matches automatically, a simple but effective user interface is required both for setting some parameters, such as the threshold and the weights for the metrics, and also for correcting and validating the match and integration results. Unfortunately, most prototypes developed so far offer no or only a rudimentary user interface, except COMA [20], COMA++ [19], and Clio [17] systems. However, COMA, COMA++ and Clio have some limitations as well as addresses above.
- **Use of Match Results for Schema Integration and Providing a Comprehensive System:** Efforts in the literature are typically about algorithms and they do not consider developing complete systems for enabling interoperability. These algorithms are useful as being the base for schema matching and integration systems, but they require a large amount of manual input. Furthermore, none of these efforts considers how to use the result of schema matching for semi-automatic schema integration. Providing a system with only schema matching capabilities and not considering schema integration is not enough and limits the applicability of the system only to specific cases.

V. THE SASMINT SYSTEM

Considering the limitations of the previous work as addressed in Section IV, a system, called Semi-Automatic Schema Matching and INTegration (SASMINT), is proposed, capable of automatically resolving naming, structural, and semantic conflicts and semi-automatically integrating relational database schemas [3]. Since user input is required after the schema matching and schema integration, it is aimed to be used by database administrators or users who have sufficient knowledge about the domain as well as database schemas.

The main components of SASMINT are shown in Fig. 6. The *Sampler Component* helps the user to identify the appropriate weight for each metric and algorithm used in the schema matching. The *Graph Representation Component* of SASMINT is responsible for representing schemas in the graph format. It uses JGraph [23] for graph visualization and JGraphT [24] for its Java graph libraries. Users interact with the system by means of the *GUI Component*. The *Schema Matching Component* matches input schemas, which are called as recipient and donor schemas, using a combination of Linguistic and Structural Matching techniques. Linguistic Matching

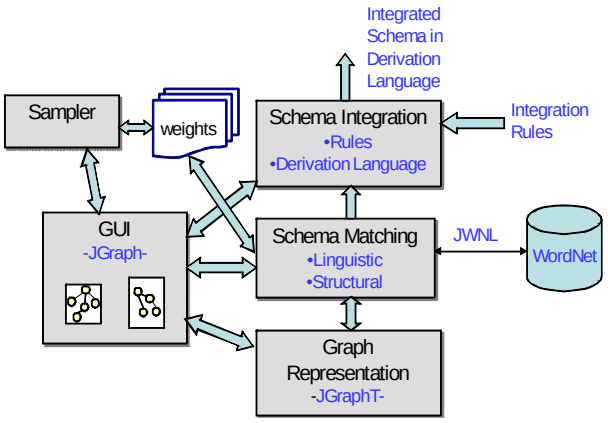

Figure 6. Components of SASMINT

benefits from the WordNet, a lexical dictionary [25]. SASMINT uses Java WordNet Library (JWNL) [26] for connecting to the WordNet. The *Schema Integration Component* integrates schemas using the results of schema matching and a number of pre-defined rules and represents the integrated schema in a derivation language, as explained in Section VIII.

SASMINT has three main processing steps: *Configuration*, *Schema Matching*, and *Schema Integration*. Details of these steps are provided in the next three sections. The main flow of information in the system is as follows: First, the user assigns weights for each metric or algorithm either manually or with the help of the Sampler component. If nothing is set by the user, default is the equal weight distribution. Secondly, the user specifies a threshold value and the selection strategy. Then, he loads the recipient and donor schemas which are converted into the graph format before being displayed. After that, the user can run the Match option, which identifies similarities between two schemas. After modifying, validating, and saving the match results, the user may continue with the schema integration. The result of schema integration is shown in both graph and derivation language format and requires the final user validation.

SASMINT can be used in different types of application domains with different purposes. In all types of the integrated information management system introduced in Section II, schema matching and/or schema integration is required. Fully federated schema and global federated schema approaches involve schema integration, which also necessitates schema matching as an internal step. On the other hand, 1-to-1 schema mapping and common schema adaptation mapping approaches require schema matching in order to identify the mappings. Helping users with the automation of both schema matching and schema integration is very crucial for rapid formation of collaboration among organizations.

VI. Configuration Step of SASMINT

Configuration step is responsible for identifying the selection strategy for the results of schema matching as well as assigning weights to the metrics and algorithms used by the linguistic and structural matching components of SASMINT. The process of identifying the strategy to be used for selecting the results of schema matching has the following flow of events:

1) Setting up of a threshold value by user: The user is asked to provide a match threshold value which is used subsequently in the process. If no threshold is specified by the user, a value of 0.5 is defaulted.
2) Getting user's preference (i.e. input) on match results selection strategy: The user is asked to choose a strategy between:
    a. Selecting all matching pairs with similarity above the threshold (called as "select all above threshold")
    b. Selecting the ones with the highest similarity if there is more than one element matching another element (called as "select max above threshold").

As for the second responsibility of the configuration step, which is the assignment of weights, currently there are three ways supported by SASMINT:

1) User can use the SAMPLER component to identify the appropriate weights for Linguistic Matching metrics. The details of this process are given in the following paragraphs.
2) Users can manually assign weights.
3) In case neither (1) nor (2) are opted for, SASMINT assumes an equal weight distribution. Needless to say, this might lead to imprecise mapping results.

SASMINT implements a composite matching technique. In this approach, the Linguistic Matching process utilizes a number of metrics, combining them by means of a weighted summation. The reason behind using different metrics is due to the variety of the element names that are compared. Certain metrics perform better than others depending on the element names being matched.

Accurate matching is important in order to reduce the amount of user input and we consider appropriate distribution of weights to be a pre-requisite for accurate matching. However, assigning these weights manually is not an easy task and assistance to the user is required. For this reason, SASMINT provides a component called Sampler, whose function is to guide the user in assigning weights to the metrics used in Linguistic Matching. In Fig. 7, the operation of the Sampler Component is illustrated.

The Sampler component can work with up to five known sample pairs. Through the GUI, shown in Fig.8, provided by the Sampler component, the user has the freedom to put in a) syntactically similar pairs in case

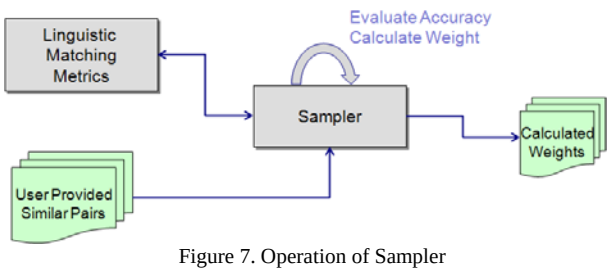

Figure 7. Operation of Sampler

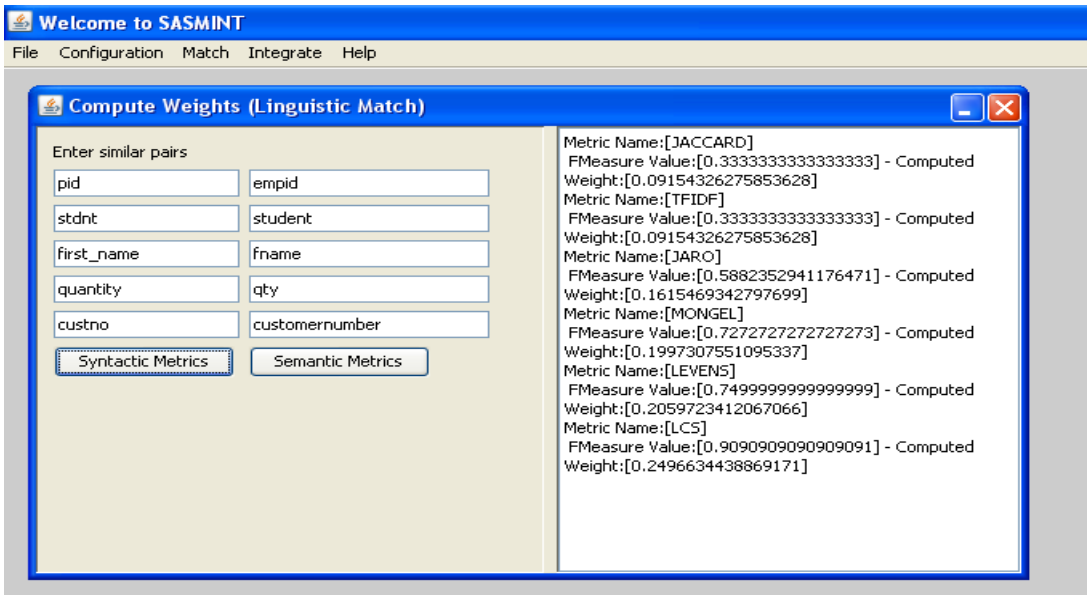

Figure 8. Use of Sampler

he/she would like the system to compute the weights of syntactic matching metrics, or b) semantically similar pairs in case it is required to compute the weights of metrics for semantic matching.

The user is expected to input these pairs into the Sampler component from his schema domain. For instance, the user might want to see how syntactic similarity metrics would perform for the pair P: ["course_credit", "credit_of_course"]. On the other hand, he might want to see how semantic similarity metrics would perform for the pair P: [“person”, “individual”].

For a given set of pairs S: {P1, P2, … , PN}, Sampler runs syntactic or semantic metrics for each given pair P in S, and determines their calculated similarities. The outcome of the calculated similarity for a Pair P is a value between 0 and 1. After the computation of the similarity values, the Sampler measures the accuracy level of each metric using F-measure. F-measure is a combination of precision and recall from the information retrieval domain [27] and used in different areas for calculating the accuracy. Using the following formula, the Sampler calculates the weight for each metric; 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’.

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

As the last step of the weight computation and assignment process, the calculated weights of metrics are presented to the user. The user has the option of accepting and directly using the proposed weights, or modifying them and feeding them back to the system for usage. An example of the usage of the Sampler component is given in Fig. 8:

VII. Schema Matching Step of SASMINT

Schema matching aims at finding all correspondences between elements of two schemas. SASMINT focuses on the schema level matching, utilizing element and structure level information. Furthermore, SASMINT exploits a combination of automatic schema matching techniques for resolving syntactic, semantic, and structural heterogeneities. Considering a single criterion (e.g., name matching) is unlikely to be successful for achieving high match accuracy for a large variety of schemas. As a consequence, it is necessary to combine and utilize multiple techniques at the same time. For this purpose, SASMINT combines the results of several independently executed linguistic and structure match algorithms.

Schema matching in SASMINT consists of the *preparation*, *comparison*, and *result generation and validation steps*, as detailed below. Fig. 9 shows an overall view of the steps of schema matching.

A. Preparation Step of Schema Matching

The Preparation step deals with the translation of source schemas defined in the typical Data Definition Language (DDL) of its Database Management System (DBMS) into a common representation format. The Directed Acyclic Graph (DAG) format with labeled edges has been chosen for this purpose, considering that it provides a balanced format among other alternatives supporting the representation of a relational schema, an object-oriented schema, etc. as a graph.

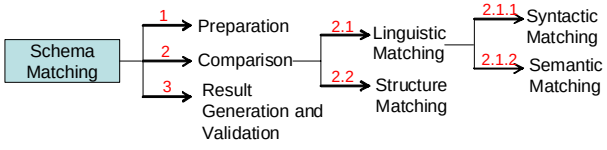

Figure 9. Steps of Schema Matching

Preparation step of SASMINT, shown in Fig. 10, works as follows: User can load the recipient schema from a database or from previously persisted XML files, in case of which schemas are already in graph format. He can load the donor schema from a database. When he chooses to load the schemas from a database, the system connects to the database using the related Java Database Connectivity (JDBC) driver, gets the metadata information (e.g. tables and columns in relational databases), represents the metadata in graph format by means of JGraphT, and finally using JGraph visualizes and displays the graphs corresponding to the schemas.

B. Comparison Step of Schema Matching

A key step of SASMINT in the schema matching process is the Comparison step, which identifies the likely matches between two schemas by resolving syntactic, semantic, and structural heterogeneities. SASMINT uses a number of algorithms from the Natural Language Processing (NLP) and Graph Theory. The Comparison step consists of two types of matching: Linguistic and Structure, detailed below.

Most of the time, element names are represented differently in different schemas, and thus before the matching process, they need to be brought into a common representation. This sub-step of SASMINT called pre-processing and involves the operations shown in Fig. 11. In tokenization and word separation operation, strings containing multiple words are split into list of words. For instance, “First Name” is split into “First” and “Name”. Stop words, such as “of” and “the” as well as some special characters, such as “/” and “-” are removed from names. Furthermore, abbreviations are expanded and lemmatization is used to bring different forms of the same word into a common form.

1) Linguistic Matching

Linguistic matching considers only the names of schema elements and results in a value between 0 and 1, for pairs of element names from the two schemas. Variety of algorithms or metrics from the NLP research field is applied to identify the syntactic and semantic similarities.

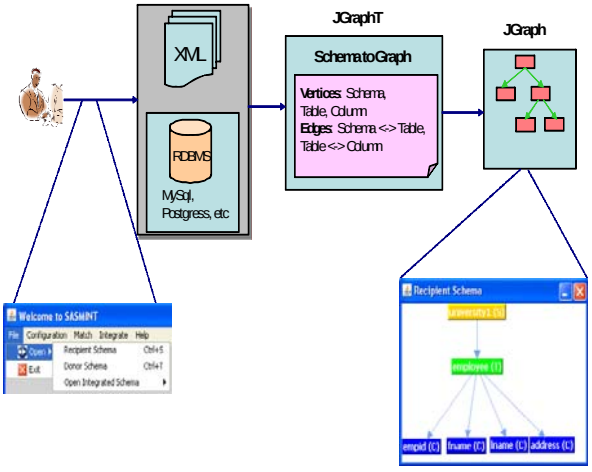

Figure 10. Preparation Step

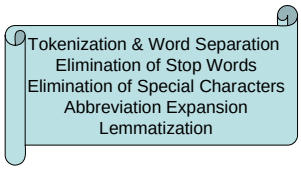

Figure 11. Pre-processing operations

In order to compare element names from two schemas, node names from the graph representation of these schemas are put into two separate lists. After preprocessing names, syntactic and semantic match algorithms are applied to each pair and then the results are combined for the final value of Linguistic Matching.

Syntactic Similarity

There are large numbers of string distance and similarity algorithms (also called here as metric) from the Natural Language Processing communities.

Unlike other approaches to schema matching, which use only one metric for syntactic similarity, SASMINT uses a combination of several main syntactic similarity metrics for comparing two character strings syntactically. These metrics can be classified as string-based and token-based. *String-based* metrics consider strings as adjacent sequences and do not divide multi-word strings into a set of single strings. However, *token-based* metrics view strings as unordered sets of tokens. As for the string-based metric, SASMINT uses *Levenshtein Distance (Edit Distance)* [28], *Monge-Elkan Distance* [29], *Jaro* [30], and *Longest Common Substring (LCS)* metrics. As for the token-based metric, it utilizes *TF*IDF (Term Frequency*Inverse Document Frequency)* [31] and *Jaccard Similarity* [32] metrics. Considering that each metric is suitable for a different type of string, SASMINT can be used for more types of strings than previous approaches.

SASMINT uses a combination of these metrics to obtain more accurate results. Metrics are combined by means of a weighted summation using the following formula:

$$sim_W(a,b)=w_{lv}*sm_{lv}(a,b)+w_{me}*sm_{me}(a,b)+w_{jr}*sm_{jr}(a,b)+w_{jc}*sm_{jc}(a,b)+w_{tf}*sm_{tf}(a,b)+w_{lc}*sm_{lc}(a,b) \quad (2)$$

where ‘lv’ stands for Levenstein, ‘me’ for Monge-Elkan, ‘jr’ for Jaro, ‘jc’ for Jaccard, ‘tf’ for TF-IDF, and ‘lc’ for Longest Common Substring.

Another contribution of SASMINT is its *recursive weighted metric*. This metric is aimed for element names containing more than one token. Depending on whether the names contain one or more tokens, the user can choose between the weighted and recursive weighted metric. Given two strings a and b that are tokenized into $a=s_1,s_2,..s_l$ and $b=t_1,t_2,..t_m$, the recursive weighted metric is calculated as follows:

$$sim(a,b)=\frac{1}{2l}\sum_{i=1}^{l}\max_{j=1}^{m} sim_W(a_i,b_j)+\frac{1}{2m}\sum_{j=1}^{m}\max_{i=1}^{l} sim_W(a_i,b_j) \quad (3)$$

Semantic Similarity

Identifying the semantic similarity between two words or concepts has been the subject of many applications in NLP, information retrieval, and some other areas. The semantic similarity measures use variety of knowledge resources, such as WordNet [25]. WordNet is partitioned into nouns, verbs, adjectives, and adverbs, which are organized into synonym sets, each representing one underlying lexical concept. Synonym sets, called also as synset, are interlinked by different relations, such as hypernymy, hyponymy, antonymy, meronymy, holonymy, etc.

Semantic similarity algorithms from the NLP domain that SASMINT uses can be classified as path-based and gloss-based measures. Path-based measures use the path between the concepts in taxonomy of concepts. SASMINT exploits the measure of Wu and Palmer [33], which is based on the idea of calculating the shortest path between the concepts in the IS-A hierarchy of WordNet. As the base for its gloss-based measure, SASMINT uses the measure of Lesk [34]. SASMINT benefits from the gloss information provided in WordNet for calculating the gloss-based similarity.

The result of semantic similarity in SASMINT is the weighted sum of the two semantic similarity measures addressed above. Following formula is used for computing the result of semantic similarity:

$$sim_{WSemantic}(a,b)=w_{wup}*sm_{wup}(a,b)+w_{gloss}*sm_{gloss}(a,b) \quad (4)$$

where 'wup' stands for Wu and Palmer's measure and 'gloss' for the gloss-based similarity.

2) Structure Matching

In addition to linguistic differences, other types of differences that frequently occur among database schema definitions are structural differences. Structural differences are more difficult to resolve than Linguistic differences, typically requiring user input. The second activity of comparison in SASMINT is structure matching, which uses the result of linguistic matching to identify the structural similarity of two schemas represented as graphs. For the purpose of structure matching in SASMINT, a variety of graph similarity and matching algorithms from the Graph Theory and other areas like Web searching and schema matching were considered.

The first approach that structure matching in SASMINT uses is the one proposed by [35]. It is an iterative algorithm from the graph similarity research field. This algorithm is based on the idea that nodes of two graphs are similar if the neighbors of these nodes are also similar.

As the second algorithm for structure matching, SASMINT uses the structure similarity algorithm of Similarity Flooding [16]. Similarity Flooding is based on a fix point computation to calculate the structural similarity. It uses an iterative algorithm and the similarity of two elements is propagated to their adjacent elements at each iteration.

Similar to the method followed in linguistic matching, structure matching uses the weighted sum of these two structural similarity algorithms, as shown in the formula below:

$$sim_{WStructure}(a,b)=w_{blondel}*sm_{blondel}(a,b)+w_{sf}*sm_{sf}(a,b) \quad (5)$$

where 'blondel' stands for the algorithm of [35] and 'sf for the algorithm of Similarity Flooding.

C. Result Generation and Validation Step of Schema Matching

Results of the comparison step are displayed to the user by means of a GUI in order for him to modify and save them. Previous systems typically provide either a primitive or no GUI. However, a clever and flexible GUI is an indispensable part of a matching system, both because it is not possible to determine all possible matches automatically and also not all the identified matches may be correct, especially considering the existence of large amount of semantics involved in schema descriptions. An example case, for which the user input is essential, occurs for complex matches, such as *1-to-n* (one column in one schema matches one or more columns in the other schema). For this case, it is not possible to automatically decide whether a column in the first schema is a combination of *n* columns in the second schema and if so, it may not be known how to combine these *n* columns, such as using concatenation, sum, etc. In order to be more specific, suppose that schema matching system has identified a match between the "rNum" element in one schema and "roomNo" and "telNum" elements in the second schema. In this case, user is supposed to delete the match between "rNum" and "telNum" as one refers to the room number and the other refers to the telephone number. As another example, suppose that the system has identified a match between the "name" element in one schema and "fname" and "lname" elements in the second schema. In this case, user is supposed to specify that "name" is the concatenation of "fname" and "lname".

Considering the requirements addressed above, a GUI is implemented as a part of SASMINT, a screenshot of which is shown in Fig. 12. Using this GUI, user can load the recipient schema from a database or from a file (in XML-based Graph format, called SASMINT Derivation Markup Language-SDML as introduced below) and donor schema from a database, as shown in two windows titled "Recipient Schema" and "Donor Schema" in Fig. 12. After loading the schemas, he can run the match option. Results of matching are displayed in graph format, as shown in the window titled "Schema Match". The window titled "Metric Results" shows similarity results that each metric has identified for all matching pairs. User can delete incorrect matches and introduce new ones and specify which kind of operation to use for combining *n* columns in 1-to-n or n-to-1 matches, using the window titled "Integration Customizations". User can either store schema matching results or continue with the schema integration. If he chooses to save the results, an XML-based SDML format is used to persist the results in

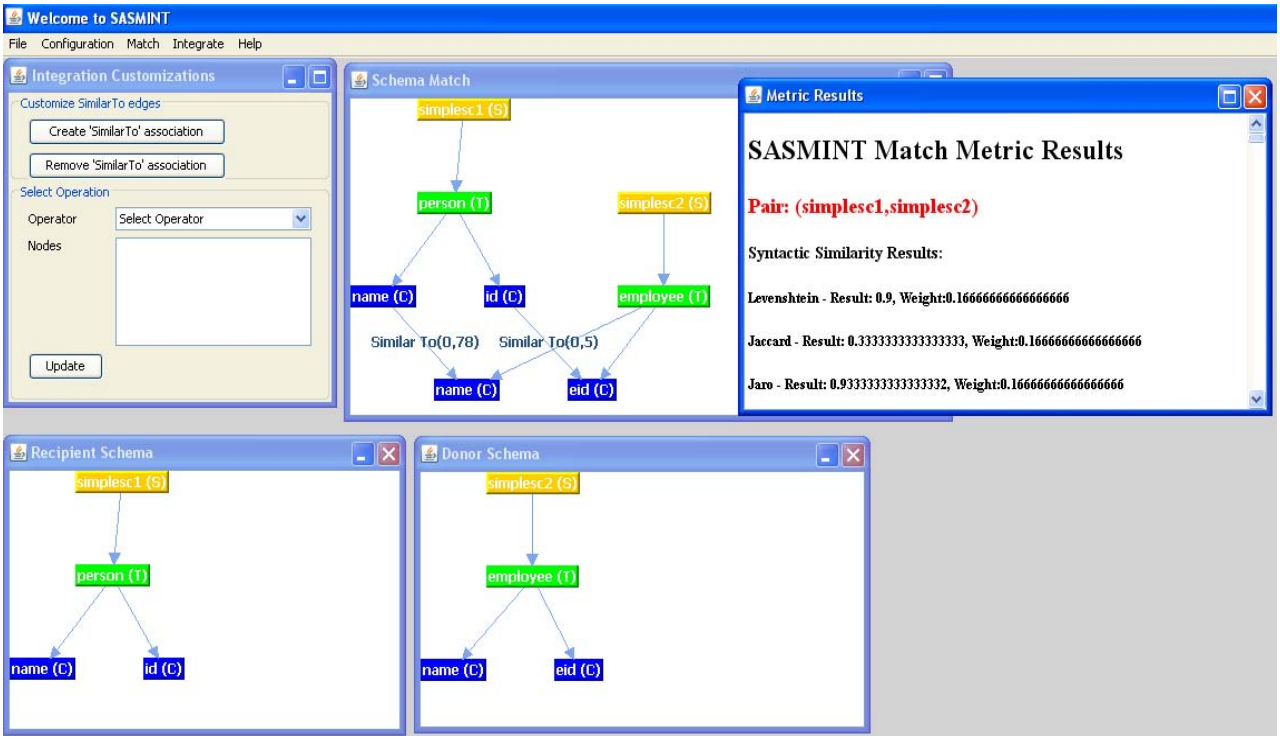

Figure 12. Screenshot of SASMINT GUI

a file. SDML is designed as a part of the SASMINT system and further explanations about it will be given in a forthcoming paper. This format is similar to other existing XML-based graph formats, such as Graph eXchange Language (GXL) [36] and GraphML [37], but it is extended to store the results of both match and integration. SDML uses a number of derivation elements, base constructs of which are explained in the following section. These derivation elements consist of tableRenameDerivation, tableUnionDerivation, tableSubtractDerivation, tableRestrictDerivation, columnRenameDerivation, columnUnionDerivation, and columnStringAdditionDerivation for storing the derivation results of schema integration. The 'columnStringAdditionDerivation' element is also used at the end of schema matching to define the special mapping rule, which means that a column in one schema is represented by the concatenation of some columns in the second schema. Content of an example XML file produced as a result of a schema matching is shown in Fig. 13. As it can be seen in the figure, a match is identified between the "fname" column of the "person" table in the first schema and the "name" column of the "employee" table in the second schema.

VIII. Schema Integration Step of SASMINT

Schema integration is a key process in many database applications. It is required in different types of integrated information management system approaches, introduced in Section II.

SASMINT facilitates schema integration by providing some semi-automatic means. After the schema matching step, users can continue with the schema integration to integrate two schemas that have been matched. SASMINT automatically generates an integrated schema, which needs the final user validation, as it is not possible to resolve all types of structural conflicts. Among different possibilities for the results of schema matching, following cases are the ones automatically handled by the schema integration component of SASMINT:

- *ColumnX (1 → 1) ColumnY*: ColumnX in the first schema matches ColumnY in the second schema.
- *ColumnX (1 → n) Column*: ColumnX in the first schema matches *n* columns of the second schema.
- *Column X (1→ 1) Table A*: ColumnX in the first schema matches Table A in the second schema.
- *Column (m → 1) ColumnY*: *m* columns of the first schema match ColumnY in the second schema.
- *Column (m → 1) Table B*: *m* columns of the first schema match Table B in the second schema.
- *TableA (1→ 1) TableB*: TableA in the first schema matches TableB in the second schema.
- *TableA (1→ n) Table*: TableA in the first schema matches *n* tables of the second schema.
- *Table A (1→ 1) Column Y*: Table A in the first schema matches Column Y in the second schema.
- *Table A (1→ n) Column*: Table A in the first schema matches *n* columns of the second schema.
- *Table (m → 1) TableB*: *m* tables of the first schema match TableB in the second schema.
- *Table (m→ n) Table*: *m* tables of the first schema match *n* tables of the second schema

Considering different conflicts to be resolved, a number of rules for integrating relational schemas have been defined for SASMINT. In order to detect integration points automatically, these rules operate on the types of

```
<graph:sgraph xmlns:grap="http://namespaces.sasmint.org/2007/04/GraphModel">
  <graph:snode grap:id="urn:sasmint:schema:schema1" grap:name="schema1" grap:type="SCHEMA"/>
  <graph:snode grap:id="urn:sasmint:table:schema1:person" grap:name="person"
   grap:schema="schema1" grap:type="TABLE"/>
  <graph:snode grap:id="urn:sasmint:column:schema1:person:fname" grap:name="fname"
   grap:schema="schema1" grap:table="person" grap:type="COLUMN"/>
  <graph:snode grap:id="urn:sasmint:schema:schema2" grap:name="schema2" grap:type="SCHEMA"/>
  <graph:snode grap:id="urn:sasmint:table:schema2:employee" grap:name="employee"
   grap:schema="schema2" grap:type="TABLE"/>
  <graph:snode grap:id="urn:sasmint:column:schema2:employee:name" grap:name="name"
   grap:schema="schema2" grap:table="employee" grap:type="COLUMN"/>
  <graph:sedge grap:id="urn:sasmint:hastable:1" grap:sourceNodeId="urn:sasmint:schema:schema1"
   grap:targetNodeId="urn:sasmint:table:schema1:person" grap:type="HASTABLE"/>
  <graph:sedge grap:id="urn:sasmint:hascolumn:2"
   grap:sourceNodeId="urn:sasmint:table:schema1:person"
   grap:targetNodeId="urn:sasmint:column:schema1:person:fname" grap:type="HASCOLUMN"/>
  <graph:sedge grap:id="urn:sasmint:hastable:3" grap:sourceNodeId="urn:sasmint:schema:schema2"
   grap:targetNodeId="urn:sasmint:table:schema2:employee" grap:type="HASTABLE"/>
  <graph:sedge grap:id="urn:sasmint:hascolumn:4"
   grap:sourceNodeId="urn:sasmint:table:schema2:employee"
   grap:targetNodeId="urn:sasmint:column:schema2:employee:name" grap:type="HASCOLUMN"/>
  <graph:sedge grap:id="urn:sasmint:similarTo:abacfd16-a04f-45ed-a3ef-98d795afce11"
   grap:sourceNodeId="urn:sasmint:column:schema1:person:fname"
   grap:targetNodeId="urn:sasmint:column:schema2:employee:name" grap:type="SIMILARTO">
   <graph:similarity>0.567814192677258</grap:similarity>
  </grap:sedge>
```

Figure 13. Result of Schema Matching in XML format

match results listed above. The rules identify which tables and columns need to be inserted in the resulting schema and how they need to be combined in order to generate an integrated schema that can represent all the elements of participating schemas. If SASMINT is extended to work with types of schemas other than relational, similar rules can be defined for these types also. Details about these rules are the subject of a forthcoming paper and thus we will not give further information about them.

The Schema Integration component of SASMINT uses a derivation language for representing integrated schemas. A formal representation of the derivation language constructs, a variation of PEER derivation language [38], is given in [3]. There are two types of derivation for relational schemas: Table and Column Derivation. Table derivation consists of derivations of type "Table Rename", "Table Union", "Table Subtract", and "Table Restrict". On the other hand, column derivation comprises 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 SASMINT. Brief explanations about all derivation types are provided below:

- *Table Rename* derivation is used when a new table is generated in the integrated schema by renaming a table in one of the input schemas (recipient and donor schemas).
  Example: FacultyMember@IntSchema = Faculty@S1.
- *Table Union* derivation is used to state that a newly generated table in the integrated schema is the union of two or more tables from the input schemas.
  Example: Department@IntSchema = union (Department@S1 , Department@S2).
- *Table Subtract* derivation is used to specify that a table in the integrated schema is constructed by subtracting a table from another table in one of the input schemas.
  Example: EngineeringDepartments@IntSchema = subtract(Departments@S1, NonEngineeringDepartments@S1).
- *Table Restrict* derivation is used to specify that a table in the integrated schema is generated by applying a restriction to a table in one of the input schemas.
  Example: SuccessfulStudents@IntSchema = restrict (Students@S1,[gpa > 2.0]).
- *Column Rename* derivation is used when a new column is generated in the integrated schema by renaming a column in one of the input schemas.
  Example: start@Time@IntSchema = start@Time@S2.
- *Column Union* derivation is used to specify that a newly generated column of the integrated schema is the union of two or more columns of the input schemas.
  Example: dptname@Department@IntSchema = {dname@Department@S1, dname@Department@S2}.
- *Column Extraction* derivation is used to specify that a column of one of the input schemas equals to two or more columns of the other input schema, combined by an operator, such as arithmetic and string operator. Currently, *columnStringAdditionDerivation* is supported, which is used to specify that a column in one schema equals to the concatenation of two or more columns in the other schema.
  Example: name@Student@IntSchema = fname@Student@S1 + lname@Student@S1.

Using the automatic schema integration rules, an integrated schema is proposed to the user. User can modify/save the result, which is stored in XML format

using the SDML. SDML uses the derivation constructs defined above as the base.

IX. EXAMPLE CASE OF SASMINT

SASMINT is a generic system and can be applied to different types of Integrated Information Management Systems as introduced in Section II, for the purpose of semi-automatic schema matching and/or schema integration. We provide in this section through a small example some discussions over the application of SASMINT.

Fig. 14 shows parts of two university schemas that also include some foreign keys (FK). In order to match and then integrate these two schemas, we ran SASMINT with the threshold value of 0.5 and selection criteria set as "select max above threshold".

SASMINT could identify all the following similar pairs correctly, namely (course, academic_course), (course_id, academic_course_id), (course_provider, academic_course_provider), (department, department), (department_id, department_id), and (dept_name, dept_name) from Shema-1 and Schema-2 are matched correctly, except for those shown in Fig. 15 that were not identified correctly but that for example a human database expert may discover through investigation of these two schemas. The element pairs shown in Fig. 15 are of two categories: either those that SASMINT missed to identify (thus representing the false negative), or those that SASMINT found as similar while actually they were not (thus representing the false positives). These cases could not be correctly identified by SASMINT, mostly due to the fact that SASMINT system currently lacks some semantic relationships. For example, the semantic similarity of "university" and "academic_institution" could not be identified through the WordNet in the current processing done by SASMINT. Furthermore, although they have different meanings, the "university" and "university_student" were identified as similar, due to their partial overlap in names, as well as their structure, while they are not correct matches.

Especially considering such semantic issues, this example indicates that a fully automatic schema matching system is not the right approach for integration of heterogeneous schemas, rather the semi-automated approach of SASMINT is suitable, that is accompanied by a sophisticated GUI to support users with their modification of the match results. Furthermore, after saving the modified matched results of the two schemas, the schema integration process of SASMINT can be started by user.

Schema-1
course (course_id, course_name, course_provider (FK))
department (department_id, dept_name, faculty_ref (FK))
faculty (faculty_id, faculty_name, university_ref (FK))
university (university_id, university_name)

Schema-2
academic_course (academic_course_id, academic_course_crdt, academic_course_provider (FK))
department (deptartment_id, dept_name, university_ref (FK))
university_student (university_student_id, name)
academic_institution (academic_institution_id, academic_institution_name)

Figure 14. Schemas from University Domain

False negatives
university ⇔academic_institution
university_id@university ⇔ academic_institution_id@academic_institution
university_name@university ⇔ academic_institution_name@academic_institution

False positives
university ⇔ university_student
university_name@university ⇔ name@university_student)

Figure 15. Missed or Incorrectly Identified Matches

Without showing the details about the derivation (for simplicity reasons), Fig. 16 represents the integrated schema generated by SASMINT for this example case. This integrated schema is complete and almost minimal. In other words, this schema covers all elements of the two schemas, while containing no redundancy except for the "university_ref" column of the "department" table, which is not incorrect, but not required in a minimal integrated schema.

We have carried out many similar experiments using different schemas in order to evaluate the performance of both the schema matching and the schema integration processes of SASMINT, of which the results are the subject of a forthcoming paper. The results of all these experiments have shown that SASMINT can achieve high percentage of accuracy, (about 75 to 85 %) with its schema matching process, and can generate complete and about 99% minimal schemas with its schema integration process. At present, the most important difficulty is in identifying some semantics involved in each schema. Currently, as a generic tool, SASMINT uses the domain independent WordNet for identifying semantic similarities. Nevertheless, WordNet does not contain domain specific semantic relationships. As a future work, in addition to using the WordNet, domain specific ontology will also be integrated to the SASMINT system, so that more types of semantic relationships can be identified, and thus the automated process of SASMINT can generate more accurate results.

Current experiments have also shown that SASMINT's GUI is very useful and makes the interaction of domain experts with the system straightforward and effective. Furthermore, the SDML format used for saving the results of both matching and integration is valuable. This format enables results to be interpreted and used by other systems for further processes, for example for the purpose of federated query processing. Moreover, it has a human-readable format that makes it very easy for the user to understand and modify the results.

Integrated Schema
course (course_id, course_name, academic_course_crdt, course_provider (FK))
department (department_id, dept_name, faculty_ref (FK), university_ref (FK))
faculty (faculty_id, faculty_name, university_ref (FK))
university (university_id, university_name)
university_student (university_student_id, name)

Figure 16. Resulting Integrated Schema

X. Conclusion

With the increasing number of Collaborative Networks, the need for an infrastructure supporting data sharing in such networks has become clear. Schema matching and integration correspond to the key components of this infrastructure. Since carrying out these tasks manually is error-prone and time consuming, some automatic mechanisms are required. This paper introduces the SASMINT system, which enables semi-automatic schema matching and integration by combining a number of syntactic, semantic, and structural similarity algorithms from the NLP and Graph Theory domains. SAMINT uses a weighted sum of different metrics or algorithms in order to be applicable for different types of strings. It is possible to semi-automatically identify the appropriate weight for each metric by means of SASMINT's Sampler tool. SASMINT provides an effective GUI for users to modify and accept match and integration results. Furthermore, utilizing the result of schema matching for schema integration and defining a set of rules for automatic integration as well as a derivation language for representing the results of both matching and integration are other contributions of the SASMINT system.

References

[1] L.M. Camarinha-Matos, H. Afsarmanesh, and M. Ollus, "ECOLEAD: A Holistic Approach to Creation and Management of Dynamic Virtual Organizations", Proc. of *PRO-VE'05*, 2005.

[2] H. Afsarmanesh and L.M. Camarinha-Matos, "A Framework for Management of Virtual Organizations Breeding Environments", Proc. of *PRO-VE'05*, 2005.

[3] O. Unal and H. Afsarmanesh, "SASMINT System for Database Interoperability in Collaborative Networks", *Lecture Notes in Computer Science*, 2006. **4275/2006**, pp. 91-108.

[4] O. Unal and H. Afsarmanesh, "Using Linguistic Techniques for Schema Matching", Proc. of *International Conference on Software and Data Technologies*, 2006, Setubal, Portugal.

[5] O. Unal and H. Afsarmanesh, "Interoperability in Collaborative Network of Biodiversity Organizations", Proc. of *PRO-VE'06*, 2006, Helsinki, Finland: Springer.

[6] T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*. 2 ed. 1999, New Jersey: Prentice Hall.

[7] A. Sheth and J. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Surveys*, 1990. **22** (3), pp. 183-236.

[8] C. Batini, M. Lenzerini, and S. Navathe, "A comparative analysis of methodologies for database schema integration." *ACM Computing Surveys*, 1986. **18** (4), pp. 323-364.

[9] E. Rahm and P.A. Bernstein, "A survey of approaches to automatic schema matching", *VLDB Journal*, 2001. **10** (4), pp. 334-350.

[10] H.H. Do, S. Melnik, and E. Rahm, "Comparison of Schema Matching Evaluations", Proc. of *Web, Web-Services, and Database Systems 2002*, 2002.

[11] S. Spaccapietra, C. Parent, and Y. Dupont, "Model Independent Assertions for Integration of Heterogeneous Schemas", *VLDB Journal*, 1992. **1** (1), pp. 81-126.

[12] F. Tuijnman and H. Afsarmanesh, "Management of Shared Data in Federated Cooperative PEER Environment", *International Journal of Intelligent and Cooperative Information Systems (IJICIS)*, 1993. **2** (4), pp. 451-473.

[13] Y. Arens, C.A. Knoblock, and W.-M. Shen, "Query Reformulation for Dynamic Information Integration", *Journal of Intelligent Information Systems*, 1996. **6** (2/3), pp. 99-130.

[14] W. Li and C. Clifton, "SEMINT: A tool for identifying attribute correspondence in heterogeneous databases using neural networks", *Journal of Data and Knowledge Engineering*, 2000. **33** (1), pp. 49-84.

[15] J. Madhavan, P.A. Bernstein, and E. Rahm, "Generic Schema Matching with Cupid", Proc. of *Very Large Data Bases*, 2001.

[16] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching", Proc. of *International Conference on Data Engineering*, 2002.

[17] R.J. Miller, L.M. Haas, and M.A. Hernandez, "Schema Mapping as Query Discovery", Proc. of *Very Large Data Bases*, 2000.

[18] F. Giunchiglia, P. Shvaiko, and M. Yatskevich, "S-match: an algorithm and an implementation of semantic matching", Proc. of *ESWS*, 2004.

[19] D. Aumueller, H.H. Do, S. Massmann, and E. Rahm, "Schema and ontology matching with COMA++", Proc. of *SIGMOD*, 2005.

[20] H.H. Do and E. Rahm, "COMA - A System for Flexible Combination of Schema Matching Approaches", Proc. of *Very Large Data Bases*, 2002.

[21] D. Beneventano and S. Bergamaschi, "The MOMIS methodology for integrating heterogeneous data sources", Proc. of *IFIP Congress Topical Sessions*, 2004.

[22] K. Saleem, Z. Bellahsene, and E. Hunt, "PORSCHE: Performance ORiented SCHEma mediation", *Accepted for publication in Information Systems Journal (Elsevier)*, 2008.

[23] JGraph, http://www.jgraph.com/, 2008.

[24] JGraphT, http://jgrapht.sourceforge.net/, 2008.

[25] C. Fellbaum, *An Electronic Lexical Database*. 1998, Cambridge: MIT press.

[26] JWNL, http://jwordnet.sourceforge.net/, 2008.

[27] C.J.v. Rijsbergen, *Information Retrieval*. 1979: Butterworth.

[28] V.I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals", *Cybernetics and Control Theory*, 1966. **10** (8), pp. 707-710.

[29] A.E. Monge and C. Elkan, "The Field Matching Problem: Algorithms and Applications." Proc. of *Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[30] M.A. Jaro, "Probabilistic Linkage of Large Public Health Data Files", *Statistics in Medicine*, 1995. **14**, pp. 491-498.

[31] G. Salton and C.S. Yang, "On the specification of term values in automatic indexing." *Journal of Documentation*, 1973. **29**, pp. 351-372.

[32] P. Jaccard, "The distribution of flora in the alpine zone", *The New Phytologist*, 1912. **11** (2), pp. 37-50.

[33] Z. Wu and M. Palmer, "Verb Semantics and Lexical Selection", Proc. of *32nd Annual Meeting of the Association for Computational Linguistics.*, 1994.

[34] M. Lesk, "Automatic Sense Disambiguation Using Machine Readable Dictionaries: How to Tell a Pine Code from an Ice Cream Cone", Proc. of *5th International Conference on Systems Documentation*, 1986, Toronto, Ontario, Canada.

[35] V.D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P.V. Dooren, "A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching." *SIAM Review*, 2004. **46** (4), pp. 647-666.
[36] Graph eXchange Language (GXL), http://www.gupro.de/GXL/, 2008.
[37] GraphML, http://graphml.graphdrawing.org/, 2008.
[38] H. Afsarmanesh, M. Wiedijk, F. Tuijnman, M. Bergman, and P. Trenning., The PEER Information Management Language User Manual. 1994, Dept. of Computer Systems, University of Amsterdam.

Mrs. 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 at the Computer Science Department of the faculty of Science of the University of Amsterdam in the Netherlands. At this faculty, she is also the director of the COLNET (Collaborative Network) group. She has received her PhD in Computer Science from the University of Southern California (USC) in 1985, and her MSc degree also in Computer Science from the University of California, Los Angeles (UCLA) in 1980. Her current research focuses on the areas of Federated /Distributed Cooperative Databases, Virtual Organizations /Virtual Laboratories /Virtual Communities, Integration of Autonomous and Heterogeneous Databases, and the design and development of specialized Web-based Applications for a wide variety of domains such as Biodiversity, Manufacturing, Tele-assistance, and Distributed Control Engineering. She has directed research in more than fifteen National, European, and International projects. She has been involved in the organization and has initiated / chaired several International conferences and workshops. She has published more than 150 articles in journals, books, and refereed conference proceedings in computer science research. She has co-edited more than ten books and various issues of international Journals. She is the Dutch representative at the IFIP TC5, and a member of the IFIP WG5.3 and WG5.5.