On semi-automated matching and integration of database schemas

Ünal Karakaş, Ö.

Publication date
2010

Chapter 4

SASMINT approach

The increase in the amount of data typically stored in electronically accessible online databases and the increase in the need for collaboration and sharing of information between various communities of interest, are among the main factors that have necessitated the development of systems, which can enable sharing of data among these databases, thereby enabling integrated access to them. This chapter describes the main related research approaches that are utilized for developing systems to enable integrated access to electronic databases. Also addressed are the limitations associated with each of those approaches. Related work presented in Section 4.1 is comprised of four areas: 1) approaches focusing on integration and interoperability of databases, 2) approaches that mainly focus on database schema matching, 3) approaches focusing on database schema integration, and 4) approaches focusing on ontology matching and ontology merging. We then introduce in Section 4.2 the SASMINT approach, under which we deal with a number of open issues in the field of schema matching and integration. We introduce the SASMINT derivation language, which is devised and introduced for automatic capturing of the results of schema matching as well as for formalizing the schema integration results. Algorithms utilized in schema matching part of the SASMINT approach are described next in Section 4.2. An overview of rules for automatic generation of integrated schemas, as well as the derivation constructs that are used to represent and store the derivation history, are addressed in details in the same section. Finally, Section 4.3 concludes this chapter and emphasizes the main achievements of the SASMINT approach.

The content of this chapter constitutes materials from three published articles, which appeared in the Journal of Knowledge and Information Systems (Unal & Afsarmanesh, 2010), in the Journal of Software (Unal & Afsarmanesh, 2009) and in Lecture Notes in Computer Science (Unal & Afsarmanesh, 2006b). Earlier version of some part of the research results of this chapter appeared in the proceedings of the PRO-VE - Network-Centric Collaboration and Supporting Frameworks (Unal & Afsarmanesh, 2006a), in the proceedings of the International Conference on Software and Data Technologies (ICSOFT) (Unal & Afsarmanesh, 2006c), and in the proceedings of the Third Biennial International Conference on Advances in Information Systems (ADVIS) (Guevara-Masis et al., 2004).

4.1 Related Research Approaches

New developments in communication technologies have made accessible large amounts of data that are stored in databases geographically distributed all over the world. As identified in Chapter 3, distributed databases are typically heterogeneous, differ both in their systems and their definitions, namely containing different data models and data semantics. To support collaboration among distributed nodes, although these databases are independently created and

administered, they need to interoperate and cooperate. Furthermore, there is an ever-increasing demand to create unified databases, to support collaboration, in such a way that users can have efficient access to distributed information resources. One fundamental question that arises while dealing with autonomous heterogeneous database systems is related to the resolution of diversity in their data models and schemas, namely schemas' syntactic, semantic, and structural conflicts. This must be addressed in both global schema approaches, where schemas of participating databases need to be integrated to generate a global schema, as well as for supporting their interoperability, where correspondences among schema elements need to be identified.

A number of approaches for providing data sharing and integration among autonomous, distributed databases have been proposed, aiming at different levels of Integrated Information Management Systems, as explained in Chapter 2. We investigate these approaches by first classifying them into four categories: 1) database integration and database interoperability approaches, 2) schema matching approaches, 3) schema integration approaches, and 4) ontology matching and ontology merging approaches. The concept map shown in Figure 4.1 lays down the logical relationships between these named approaches. Schema matching is considered to be a part of (i.e. classified under) schema integration, while ontology matching/merging is a research subject, which is studied in a manner similar and with overlaps to both schema matching and schema integration. All three research subjects logically constitute a part of database integration and interoperability. In the following sections, we exemplify these four approaches. As explained in these sections, so far the proposed approaches mostly involve large amounts of manual work. They either require database designers to explicitly integrate knowledge between data sources, or provide limited automation to integrate certain specific data sources. Furthermore, manual integration processes do not scale well when the number and the size of the databases increase.

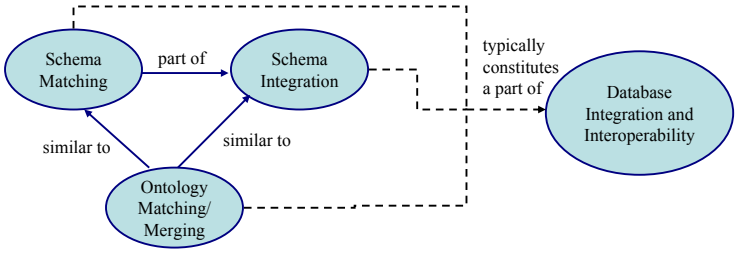


Fig. 4.1. Concept Map of Related Research Approaches

In this section, after addressing different classes of related research in sections 4.1.1 to 4.1.4, in section 4.1.5 a list of open issues for the area are provided, which both reflects a general analysis of related research in this area, as well as motivating our proposed SASMINT approach.

4.1.1 Database Integration and Interoperability Approaches

This category consists of approaches that have the main goal of database integration and database interoperability. Research initiatives and approaches that belong to this category date back to 1990's.

An early approach for resolving the semantic heterogeneity to enable interoperability in federated database management systems is proposed in (Hammer & Mcleod, 1993). A semantically rich object-based common model is introduced for describing the structure, constraints, and operations of the sharable exported data. As a result, before any sharing occurs, export schemas for each participating component databases are defined in terms of this model, by the domain experts, which is typically a manual process. Each component database also defines a local lexicon where the semantic information about the sharable objects is provided using the terms from a dynamic common list of the federation. Through utilizing the local lexicon as well as a semantic dictionary and a list of meta-functions supported by all participating databases, semantic heterogeneities are resolved.

The PEER is a federated information management system (Afsarmanesh et al., 1996; Tuijnman & Afsarmanesh, 1993) developed at the University of Amsterdam and rooted in the approach addressed above, as also explained in Section 2.2. However, no automation is provided in PEER for generating the federated database architectures at autonomous nodes.

The MOMIS (Mediator EnvirOnment for Multiple Information Sources) project (Beneventano & Bergamaschi, 2004; Bergamaschi et al., 2001; Bergamaschi et al., 1998) follows a semantic approach based on Description Logic (DL) (Brachman & Schmolze, 1985) to integrate heterogeneous information sources. As a common data model, MOMIS uses ODL_{I^3} (Bergamaschi et al., 1998), based on the ODL, which is the standard data manipulation language of the Object Oriented Database Management Systems. In this approach, wrappers need to be defined for source schemas to translate them into ODL_{I^3} before integrating them into the global schema. More information about the MOMIS system is given in Section 4.1.3 about schema integration approaches.

The InfoSleuth project (Bayardo et al., 1997) extends the ideas developed in the Carnot project (Huhns et al., 1992) and uses the agent technology, domain ontologies, brokerage, and the Internet computing, in order to achieve interoperability. When a data node joins the system for the first time, it advertises to the Broker Agent the concepts in the common ontologies that it can understand. Users can then query the system by formulating the query in any of the common ontologies. Resource Agents handle transformations between data schemas and ontologies. Queries expressed in ontology terms need to be translated into database schema terms and the results of queries need to be translated into terms that the requesting agent can understand. Mapping information is necessary for this task. However, one limitation of the system is that the mapping information needs to be created by domain experts experienced with the system during the agent installation time.

In the SIMS (Services and Information Management for decision Systems) Project (Arens et al., 1996), in order to provide access to heterogeneous and distributed databases, first a common domain model is created using the Loom knowledge representation language (Gregor, 1988). When an information source decides to join the SIMS system, first its contents need to be modeled and then the concepts in information source model need to be correlated with (i.e. associated with) the corresponding concepts of the domain model. This requirement is one major drawback of the SIMS project as it requires manual effort. Another limitation is that the user is assumed to be familiar with Loom as he is required to formulate the queries as Loom statements. The SIMS translates the query from the Loom statement into the query language of the information source and executes it. SIMS is an intermediate layer between data sources and users.

Observer system (Mena et al., 2000) is based on the idea of using multiple pre-existing domain ontologies expressed in the DL and follows a mediated approach. The system also uses some pre-defined ontologies such as WordNet and subsets of the Bibliographic-Data

ontology. As the main limitation, the Observer system requires manual processing for two tasks: 1) The content of each data repository is described by one or more ontology, 2) One-to-one mappings between the ontologies need to be defined to enable query processing. Furthermore, there is an assumption that users are familiar with the structure of the ontologies and can navigate through the ontologies and construct their queries by means of a GUI.

TSIMMIS (The Stanford-IBM Manager of Multiple Information Sources) (Garcia-Molina et al., 1997) provides tools to facilitate the rapid integration of heterogeneous information sources. It uses Object Exchange Model (OEM) (Papakonstantinou et al., 1995) to represent data sources and the Mediator Specification Language (MSL) (Papakonstantinou et al., 1996) to encode the semantic knowledge as a set of rules. Human intervention is required for these processes.

COIN (COntext INterchange) project (Goh et al., 1999) aims at integrating data sources by providing semantic interoperability between them. It uses a domain model (shared vocabulary) that defines the application domain of data sources. However, one limitation is that once defined, each data source needs to use this model. COIN performs data integration based on logical axioms. A context mediator is used for querying to reconcile potential conflicts between the data source information expressed as axioms. Another drawback of COIN is that it only uses semantics of data items and does not consider schema level conflicts.

A summary of the above mentioned approaches aimed at solving the problem of data sharing among distributed and heterogeneous systems is given in Table 4.1. Considering the multi-database architectures described in Chapter 2, the approaches mentioned in this section follow either a global schema approach or an interoperability approach. In both approaches, schema matching plays an important role, although these approaches skip the automation of schema matching.

4.1.2 Schema Matching Approaches

While on one hand enabling the sharing among distributed, heterogeneous, and autonomous data sources has been an important topic in the database research for some decades, on the other hand schema matching has usually been considered as a separate challenging problem. Therefore, schema matching related challenges have been addressed by a number of other research and development projects. In these projects, a great deal of efforts has been spent on studying of and devising ways for increasing the degree of automation of matching process.

However, all these projects are limited in the solutions that they can provide, namely requiring substantial amounts of manual work. Furthermore, their usage of linguistic techniques, which are needed in order to increase the overall accuracy of the schema matching results, is not effective either. Another limitation of these projects is that semi-automatic schema matching is not combined with other interoperability requirements, such as schema integration and distributed query processing. Typically, the provided solutions focus only on some specific parts of the problem and fail to provide a comprehensive solution. These solutions and the associated shortfalls are discussed below.

SEMINT (SEMantic INTegrator) (Li & Clifton, 2000) system utilizes both schema and instance information in order to identify mappings between relational schemas. Attributes in the first database schema are first clustered using neural networks and then similarities between the categories of attributes from the first database and the features of attributes from the second database are computed. SEMINT does not support name matching and structure matching. Furthermore, no GUI is provided within the system.

Cupid system (Madhavan et al., 2001) exploits a combination of the name and structure matchers. Schemas to be matched are represented as graphs. The first step of name matching is called the normalization step, which is not comprehensive enough to consider all normalization issues. Moreover, name matching involves a syntactic matching, which employs only one string similarity algorithm, which clearly cannot be optimal for all cases.

The COMA (COmbination of MAtching algorithms) system (Do & Rahm, 2002) provides a library of matchers that utilize elemental and structural properties of schemas. However, it does not support the pre-processing of element names. COMA++ (Aumueller et al., 2005) is built on top of COMA, by elaborating in more detail the alignment reuse operation. It provides more efficient implementations of the COMA algorithms and a sophisticated graphical user interface. Although it provides a library of different types of matchers, it does not provide assistance to users for deciding on the best matcher or combination of matchers.

Table 4.1- Database Integration and Interoperability Approaches

Project/System/Approach	Description/Aim	Multidatabase Architecture	Degree of Automation
Approach of (Hammer & Mcleod, 1993)	Resolving Semantic Heterogeneity in Federated Database Systems	Federated database	manual
PEER	Enabling information sharing among autonomous and heterogeneous nodes	Federated database	manual
MOMIS	Generating integration of heterogeneous information sources	Global schema	semi-automatic
InfoSleuth	Establishing ontology-based interoperability	Interoperability	semi-automatic
SIMS	Providing access to distributed and heterogeneous databases	Global Ontology	manual
Observer	Introducing ontology-based approach for query processing in global information systems	Interoperability - Mediator system	manual
TSIMMIS	Enabling integration of heterogeneous information sources	Interoperability - Mediator system	semi-automatic
COIN	Enabling integration of heterogeneous information sources	Interoperability - Mediator system	semi-automatic

The Similarity Flooding (Melnik et al., 2002) approach 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 applies the knowledge of edge and node semantics, nor it provides a GUI.

Similarly, (Wang et al., 2004) borrows the string similarity implementation of Similarity Flooding. This approach, although extends the structure matching part of the Similarity Flooding, suffers from the same limitations as Similarity Flooding.

Clio (Miller et al., 2000) generates alternative mappings as SQL view definitions based on the value correspondences defined by the user. Since the value correspondences are defined by the user, no linguistic matching techniques are used and substantial manual work is required.

PROTOPLASM (PROTOtype PLAtform for Schema Matching) (Bernstein et al., 2004) aims at providing a customizable industrial strength matching system that can match real world schemas. PROTOPLASM follows a composite matcher approach. Inspired from the COMA system, PROTOPLASM is based on the algorithms of CUPID and Similarity Flooding and thus has the same limitations as these two systems have. BizTalk Mapper (Biztalk, 2010) is used as the GUI for manipulating the mappings.

S-Match is a schema-based schema matching system (Giunchiglia et al., 2004). It accepts two graph structures as input and identifies semantic relationships between their nodes. Its main goal is the semantic matching and it exploits a number of element level and structure level match techniques in this process. Structure level match uses propositional satisfiability. Unlike many other matching systems, the result is not in the range [0,1] but it represents the identified semantic relations using the terms of *equivalence*, *more general*, *less general*, *mismatch*, and *overlapping*. Furthermore, it does not provide any GUI.

The results of several other schema matching efforts have been published in (Bernstein et al., 2004; Embley et al., 2004; Rahm et al., 2004). The main focus of all the work reported there is on matching large schemas or extensibility of the developed system. Moreover, some research has been carried out focused on the issue of uncertainty (Gal, 2006; Gal, 2007), which exists especially because of semantic differences. However, they share similar problems with most previous efforts mentioned above. Namely, they either require much manual work rendering the system ineffective to use, or if they use linguistic techniques, it is typically a limited use of these techniques. Table 4.2 shows an overview of schema matching systems and prototypes addressed in the previous paragraphs, also denoting the schema types supported by each system.

4.1.3 Schema Integration Approaches

One specific application of the schema matching approaches is for merging a set of schemas into a single global schema (Batini et al., 1986; Elmagarmid & Pu, 1990; Sheth & Larson, 1990). This problem has been studied since the early 1980s and is applied to building a common database system comprising several distinct databases, and in designing an integrated schema for a database from the local schemas supplied by several user databases. The integration process requires establishing semantic correspondences between the component schemas, and then using the matching results to merge schema elements (Batini et al., 1986; Pottinger & Bernstein, 2003). However, establishment of semantic correspondences is handled manually in most previous approaches.

Table 4.2- Schema Matching Systems

Project/System	Matchers Used		Schema Type	Internal Representation	Result	GUI
	Instance-Based	Schema-Based				
SEMINT	Neural Network	constraint-based	SQL	attribute-based	similarity in [0,1]	no
Cupid	-	string-based, synonyms, thesauri lookup, tree matching	SQL and XDR	tree	similarity in [0,1]	no
COMA / COMA++	-	string-based, synonyms, thesauri lookup, type use, reuse, tree matching	SQL, XDR, XSD, OWL	graph	similarity in [0,1]	yes
SF	-	string-based, fix point computation	SQL, graph	graph	similarity in [0,1]	no
Clio	Naïve Bayes	string-based	SQL, XSD	graph	mapping query	yes
Protoplasm	-	string-based, synonyms, thesauri lookup, path, fix point computation	XSD, SQL, ODMG	graph	similarity in [0,1]	Yes
S-Match	-	string-based, sense-based, gloss-based, propositional satisfiability	XSD	graph	semantic relations	no

As for schema integration, a number of systems or approaches have been introduced in the database literature. MOMIS has a component responsible for schema integration, called Artemis. In order to avoid confusion, from here on we will refer to it as the MOMIS-Artemis system, instead of its Artemis component. MOMIS-Artemis requires that all elements of schemas are annotated by the database designer manually using the appropriate meanings in

the WordNet lexical database. Common thesaurus is generated by MOMIS-Artemis describing inter and intra schema relationships. It uses the schema and annotation information. Furthermore, it allows users to define any other relationships manually. DL is used to infer new relationships from the existing ones. Similar classes are identified by using the relationships defined in the common thesaurus. For this purpose, affinity coefficients of two classes are identified based on their names and their attributes, corresponding to name and structural affinity coefficients. Name and structural affinity only consider semantic relationships between the names. These coefficients are then combined into global affinity coefficients. Hierarchical clustering uses global affinity coefficients to classify classes. For each cluster, a global class with attributes is generated. Mappings between the local and global attributes are defined in a mapping table. MOMIS-Artemis requires a database specialist to assist with the integration process at each phase of integration.

One approach for schema merging that is introduced as part of a prototype system, is called Rondo (Melnik et al., 2003). Rondo is developed as a model management tool. Since the focus of Rondo is simplicity, it is developed to show how a number of model management operations can be supported by means of a model management tool. Rondo represents models as directed labeled graphs and manipulates models and mappings among models by providing a number of operators, such as match, delete, traverse, extract, and invert. For the Match operator, Similarity Flooding is used, and thus it suffers the same limitations as those of the Similarity Flooding. The Merge operator is based on heuristics to automate the merge process. However, its automation is limited and requires human intervention. For example, it cannot automatically satisfy the condition that merged schema is at least as expressive as the input models.

Pottinger and Bernstein (Pottinger & Bernstein, 2003) propose another algorithm for merging models. The authors define a meta-meta-model called Vanilla. They aim to support models in both Vanilla and any other meta-meta-model. However, a limitation is that they assume that correspondences between two models to be merged are given beforehand. (Pottinger & Bernstein, 2008) extends this work with the work on both merging models and also generating view definitions by means of defining mappings between source schemas and the merged schema. Authors introduce a normal form, named Mediated Schema Normal Form (MSNF), for the mediated schemas and view definitions. Similar to the main limitation of (Pottinger & Bernstein, 2003) approach, they assume that correspondences are defined manually before their merge algorithm executes.

COMA++, introduced above among the schema matching systems, provides functionality for schema merging, but since schema matching is the main focus of COMA++, its schema merging approach is limited and it is not possible to see how the elements of merged schema relate to the local schemas, namely no mappings are defined between the merged schema and the local schemas.

A recent schema integration system is PORSCHE (Performance ORiented SCHEma mediation) (Saleem et al., 2008). PORSCHE 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. It uses tree mining and clustering techniques for calculating contextual semantics and for grouping similar labels to support performance oriented schema matching. PORSCHE follows incremental binary ladder integration. There is no GUI provided by PORSCHE.

Similar to Pottinger and Bernstein, Chiticariu, Kolaitis, and Popa (Chiticariu et al., 2008) propose another algorithm for integrating relational or XML Schemas. Besides providing an algorithm that enables generation of a single integrated schema, they also propose an

enumeration algorithm that can generate all plausible integrated schemas. Both of these algorithms require substantial amounts of user input. Moreover, it is assumed that correspondences among source schemas are specified beforehand. In their earlier work (Chiticariu et al., 2007), they show how their algorithm uses the correspondences identified by Clio and how Clio can help in generating mappings between the source schemas and the integrated schema. However, since their main focus is on schema integration, they generate the correspondences only manually using the GUI of Clio.

Although there is some work on semi-automatic schema integration, as summarized in Table 4.3, these proposed solutions are not generic. Instead of generating a comprehensive system and providing a complete solution, each work focuses on supporting certain specific subject, as represented in the second column of this table. In most cases it is assumed that correspondences among source schemas are already given as input. Furthermore, it is not clear how to further use the results of schema integration, for example for executing the query processing over the integrated schema.

Table 4.3- Schema Integration/Merging Approaches

Project/System/Approach	Main Purpose	Semi-Automatic Schema Matching Support	GUI
MOMIS-Artemis	Integration of heterogeneous information sources	Name and structural affinity based on semantic relations	Yes
Rondo	Model Management	Similarity Flooding is used for matching	Yes
Approach of (Pottinger & Bernstein, 2003)	Model Management / Model Merging	No	No
COMA ++	Schema Matching	A comprehensive library of matchers	Yes
PORSCHE	Schema Integration and Mediation	Linguistic Matching (Abbreviation and synonym tables generated by the user), tree mining for contextual semantics	No
Approach of (Chiticariu et al., 2008)	Schema Integration	No	Yes

4.1.4 Ontology Matching and Ontology Merging Approaches

Another area of research similar to schema matching and schema integration is the ontology matching and ontology merging. This has been an active research area especially with the

increasing popularity of Semantic Web, and so far a number of systems have been developed in this area, which we describe below. Since the aim of this thesis is schema matching and schema integration, only brief information about the ontology matching and merging systems is provided here and also our proposed SASMINT approach does not contribute much to this area. Ontology matching and merging systems described below are shown in Table 4.4.

The ONION (ONtology compositION) (Mitra et al., 2001) system uses a graph-oriented model for representation of ontologies. Since ontologies are translated into this model before matching process, ONION can accept ontologies represented in any ontology language. It identifies candidate matches between concepts specified in two ontologies and expects a domain expert to verify the results. It is assumed that the relationships among concepts are defined using a set of relationships with pre-defined semantics. Here, a number of heuristic matchers are used, such as the linguistic matching, the structure matching, and inference-based matching. Extensive manual effort is required for defining the relationships among concepts.

GLUE (Doan et al., 2002) requires ontologies to be represented as taxonomies, in which concepts are represented as nodes and is-a relationships between concepts are represented with edges between them. It provides a name matcher and several instance-level matchers by extending the schema matching system LSD (Doan et al., 2001). It uses machine-learning techniques. However, in order to train learners, ontologies are first mapped manually. Moreover, a set of domain synonyms and constraints are defined before any matching occurs. Thus, it requires extensive manual effort. Another extension of LSD is iMap (Dhamankar et al., 2004). Besides one-to-one matches, iMap can determine complex matches among relational schemas by using a number of machine learning matchers.

Naïve Ontology Mapping (NOM) (Ehrig & Sure, 2004) and Quick Ontology Mapping (QOM) (Ehrig & Staab, 2004) are the components of FOAM (Framework for Ontology Alignment and Mapping). FOAM is a tool that enables semi-automatic ontology alignment and mapping by using a number of similarity heuristics (Ehrig & Sure, 2005). NOM requires ontologies to be represented in RDFS format. Using a number of similarity functions, it computes similarity between each possible pairs of entities from two ontologies. QOM optimizes the NOM with an efficient mapping algorithm. However, this optimization causes the mapping quality to decrease.

MAPONTO (An et al., 2006) is developed as a plug-in for Protégé, which is an open source ontology editor and knowledge-base framework (Protege, 2010). MAPONTO is a semi-automatic tool helping users to identify mapping rules between database schemas and ontologies. It uses a generic conceptual modeling language (CML) for representing ontologies. This language contains common aspects of different ontology languages, UML, etc. It is assumed that user provides simple correspondences between a schema and ontology. Although this feature is regarded as an advantage by (An et al., 2006), the amount of required user input increases when the size of the schema and ontology grows, turning this feature to a disadvantage. Based on the user provided correspondences, MAPONTO infers complex formulas to represent semantic mappings.

Chimaera (Mcguinness et al., 2000) is a web-based tool that supports ontology merging as well as ontology testing and diagnosing. It accepts ontologies in a wide variety of languages. Its aim is to help users with these tasks by means of editing tools, where merging process is user-oriented and requires a lot of user intervention.

The PROMPT suite consists of a set of tools that have the purpose of ontology alignment, merging, and versioning (Noy & Musen, 2003). Its ontology merging tool is called as iPROMPT (Noy & Musen, 2000) and the ontology alignment tool is called as Anchor-PROMPT (Noy & Musen, 2001). iPROMPT guides the user through the merging process. Anchor-PROMPT takes as input two ontologies represented in graph format and finds correlations between the concepts of these different ontologies. The main limitation of

Anchor-PROMPT, declared by (Noy & Musen, 2001) is that it does not work well when one ontology is deep and the other ontology is shallow. PROMPT is implemented as an extension to the Protégé 2000 ontology development environment (Protege, 2010). PROMPT supports RDFS and OWL and as the underlying knowledge model, it uses Open Knowledge Base Connectivity (OKBC) (Chaudhri et al., 1998).

OntoMerge (Dou et al., 2003) is a tool that aims at ontology translation by ontology merging and automated reasoning. It supports ontologies represented in DAML or DAML+OIL and converts them into an internal representation. An ontology expert generates a merged ontology by taking the union of concepts and axioms from two source ontologies. Axioms define the relationships between instances of concepts. Experts add bridging axioms to relate terms from two ontologies. Therefore, the main limitation of OntoMerge is that it requires large amount of user involvement.

Table 4.4- Ontology Matching and Merging Approaches

Project/System/Approach	Main Purpose	Ontology Language
ONION	Ontology matching	Ontology in any language is translated into directed labeled graphs
GLUE and iMap	Ontology matching	Taxonomies
NOM and QOM	Ontology matching	RDFS
MAPONTO	Ontology mapping	CML – ontologies are translated into this language
Chimaera	Ontology merging, testing, and diagnosing	Wide variety of languages, such as OWL, RDFS, etc.
PROMPT	Ontology merging	RDFS, OWL, OKBC, and other languages
OntoMerge	Ontology merging	DAML, DAML+OIL

4.1.5 Open Issues and the Proposed Approach

As exemplified above, there have been many proposals and research prototypes within the last decades, aimed at achieving interoperability, matching, and integration of autonomous and heterogeneous databases. The need for schema matching in large number of applications has led to the development of many algorithms that to certain level semi-automatically solve the matching and integration problem. However, there are a number of open issues, which are not yet addressed sufficiently in previous work and thus required further investigation, as addressed in the four categories below:

a) Providing possibility to combine match algorithms

As specified under Section 4.1.2, there exist several well-known matching algorithms addressed in different literature. Each algorithm is suited for certain specific schema matching case. With this said, the proposed schema matching efforts tend to apply a limited number of these algorithms. However, in order to achieve high matching accuracy, research has shown (Aumueller et al., 2005; Bernstein et al., 2004) that it is necessary to combine a variety of types of algorithms, that can consider syntactic, semantic, as well as structural differences among schemas.

Several previous systems have used either hybrid or composite combinations of different match approaches. *Hybrid* approach uses multiple criteria or properties (e.g. name and data type) within a single algorithm, whereas the *composite* approach combines the results of several independently executed match algorithms. Since poor match candidates can be filtered out early in the process, the hybrid approach in general can identify better match candidates than the composite approach. However, the criteria and the order of evaluation in the hybrid approach are fixed, which makes it difficult to extend the algorithm used in this approach. On the other hand, since the composite approach uses different independent algorithms, it is easy to add or remove an algorithm to it, which makes this approach more flexible and applicable to different domains that require different types of matchers. Any schema matching approach needs to consider all these aspects before choosing between the hybrid and composite approaches.

b) Providing graphical user interface

A user-friendly interface is necessary considering the fact that not all the semantic correspondences between schema elements can be automatically identified by the system. User input is required both for specifying some parameters (such as the threshold value for similar pairs and weights for the algorithms), as well as for correcting and validating both the match results and the integration results. Unfortunately, all prototypes developed so far and mentioned in this chapter offer either none or only a rudimentary user interface, except for COMA (Do & Rahm, 2002) and Clio (Miller et al., 2000). Although these two offer a GUI, they have other limitations in the solutions that they offer for semi-automatic schema matching, which we discussed in Section 4.1. Consequently, one of the key aims of our schema matching and integration system is providing a user-friendly GUI.

c) Supporting use of match results for schema integration and providing a comprehensive approach

Efforts in the literature are mostly focused on matching algorithms and they do not consider developing complete systems enabling database integration and interoperability. These algorithms are useful as being the base for schema matching and schema integration and for developing the interoperability systems. But these efforts do not address how to use the results of schema matching for semi-automatic schema integration, which is necessary for our proposed approach, and limiting the applicability of the approach only to specific cases.

d) Supporting accuracy evaluation in comparison to other approaches

In order to assess the accuracy of any suggested approach, it is required to first generate different types of schemas covering a variety of syntactic, semantic, and structural heterogeneities and then test the approach against these schemas. It is also important to perform the same tests with other available systems or compare all approaches with certain

standard other evaluation results. However, the evaluations carried out in the literature for the related research does not typically apply the same test cases, so their published results cannot be used for our comparison.

Considering the limitations of previous work as addressed in this chapter, this thesis defines an approach, called the SASMINT, which aims at automatically resolving syntactic, structural, and semantic conflicts among relational schemas as well as efficiently integrating them and formalizing their integration. It creates a composite merge of some standards and widely accepted research algorithms that are suggested and applied for 1) natural language processing, 2) graph theory, and 3) meta-data design, to create a semi-automatic schema matching and integration methodology used for identification of mappings among elements and construction of integrated schemas, thus providing access to autonomous, distributed, and heterogeneous databases.

4.2 Proposed Approach: SASMINT

Automatic resolution of schema heterogeneity to enable semantic interoperability among networked databases is vital for collaborative networks of organizations. Therefore, due to growing increase in emerging collaborative networks in many domains, this area of research is timely and important. Integration and interoperability infrastructures to support these networks require effective mechanisms not only to interlink database schemas, but also to provide homogeneous access and integrated interface to heterogeneous distributed databases. In a network of organizations, whenever a new organization joins the network, its schema (referred here as donor schema) needs to be matched and integrated into the common integrated/federated schema of the network (referred here as recipient schema), which results in the *new extended common integrated schema*.

Research literature represents a variety of approaches for addressing these needs, as stated in the previous section. Aiming to overcome the limitations of other presented approaches, we propose the SASMINT approach (Unal & Afsarmanesh, 2006a; Unal & Afsarmanesh, 2006b; Unal & Afsarmanesh, 2006c; Unal & Afsarmanesh, 2009; Unal & Afsarmanesh, 2010). Some of the main features of the SASMINT are shown in Table 4.5. The main purpose of SASMINT is schema matching and schema integration, which is done semi-automatically. It supports a combination of schema-based matchers. Since instance data is not available all the time, SASMINT does not utilize any instance-level matchers. The main distinctive feature of our approach, when compared to other approaches in the literature is that besides suggesting a generic way to effectively identify the matches between schemas, these match results are also used for schema integration. Furthermore, SASMINT formalizes the results of schema matching and schema integration in an effective format, called as the SASMINT Derivation Markup Language (SDML).

The SASMINT approach can be used in different types of application domains, and for different purposes, as shown in Figure 4.2-a, 4.2-b, and 4.2-c:

1. *Database Federation with a Common Schema:* In some application cases, a common schema is predefined for the network of nodes and each node is required to develop mappings from its local schema to this common schema. SASMINT can help with identification of these mappings in a semi-automatic fashion. Sections 4.2.3 and 4.2.4 explain how these mappings are generated by SASMINT.

2. *Full Database Federation:* In fully federated systems, each node autonomously decides to share a part of its data with others, by defining export schemas. Then other

Table 4.5- Main features of SASMINT

Project/ System/ Approach	Main Purpose	Schema Type	Internal Repr.	Schema-based Matchers	Degree of Automation	Result	GUI
SASMINT	Schema Matching and Schema Integration	SQL DDL	DAG in SDML format	syntactic and semantic (from NLP), structure (from graph theory and from schema matching)	semi-automatic	similarity scores in [0,1]. Results represented in SDML format	yes

nodes import these schemas and integrate them with their own local schemas. For this purpose, SASMINT supports the semi-automated generation of individually integrated schemas at each node.

3. *Incremental Generation of Integrated Schema:* In this case, the aim is to incrementally generate a global schema, representing the sharable information of all participating nodes, as introduced in Chapter 2, and shown in Figure 4.2-c.

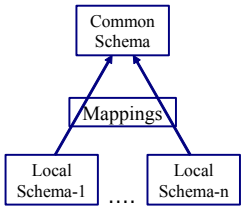


Fig. 4.2-a. Database Federation with common schema

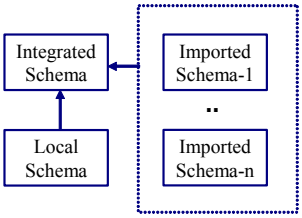


Fig. 4.2-b. Full database federation

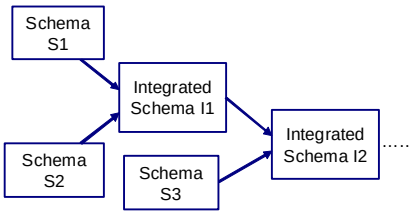


Fig. 4.2-c. Incremental generation of a global integrated schema

SASMINT achieves its goals by following the phases shown in Figure 4.3. It involves the main phases of Configuration, Automatic Schema Matching, User Modification/Validation (of match results), Schema Integration, and User Modification/Validation (of integration results).

This chapter first introduces the SDML, as it forms the base for formalizing both the results of schema matching and schema integration. More details about different phases of the SASMINT approach are provided next, starting with the configuration phase (4.2.2). Then, the automatic schema matching phase is described (4.2.3), followed by the user modification and user validation of the results generated by the automatic schema matching process (4.2.4). Details of how we use the results of schema matching for generating the integrated schema, as well as how the derivation constructs are used for defining the integrated schema are then explained in the section labeled as the schema integration phase (4.2.5). Finally, the user modification and the validation phase required on the resulted integrated schema are explained (4.2.6).

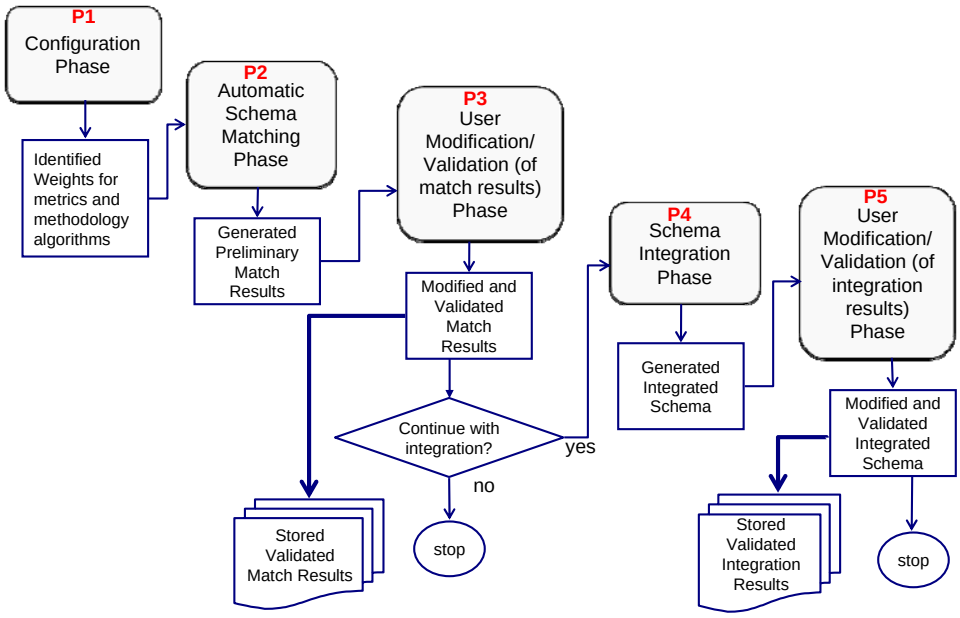


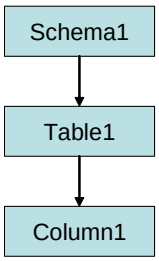
Fig. 4.3. Main Phases of SASMINT

4.2.1 SASMINT Derivation Markup Language (SDML)

In order to formally capture and store the final results of schema matching and schema integration, an XML-based SASMINT Derivation Markup Language (SDML) format is introduced and used by SASMINT. For this purpose, XML is chosen, as it provides a flexible format for storing and exchanging graphs. Furthermore, there is a wide range of tools available for parsing and querying XML. The SDML format is similar to other existing XML-based formats for representing graphs, such as the Graph eXchange Language (GXL) (Gxl, 2010) and the GraphML (Graphml, 2010). Nevertheless, the SDML is extended so that it can store both the results of the matching and those of the integration stages.

In Figure 4.4, a simple **generic schema** is represented in graph format, and its corresponding SDML representation is also presented in this figure. The root element of the SDML document is *sgraph*, which consists of two main sub-elements: *snode* and *sedge*, as explained below:

- **snode**: represents a node of the graph and contains derivation constructs as subelements. More information and examples for these derivation types are given later in this chapter. 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 comes from the name of schema, table, or column, depending on which type of schema element this node represents.




```
<?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. 4.4. A simple graph representing a generic schema and its SDML representation

- **type**: indicates whether the schema element that this node represents is of type schema, table, or column.
- **schema**: represents the name of the schema, where this node exists. This attribute is optional. If the node is of type *schema*, corresponding *snode* definition does not include the *schema* attribute.
- **table**: represents the name of the table, where this node exists. This attribute is optional. If the node is of type *table*, corresponding *snode* definition does not include the *table* attribute.

- **sedge**: represents an edge of the graph and has a sub-element called *similarity,* if this is an edge connecting two similar nodes. The *similarity* 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, which is the starting point of the edge.
  - **targetNodeId**: identifies the id of the target node, which is the end point of the edge.
  - **type**: indicates the type of the edge. The value 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.

A class diagram, corresponding to the complete features of SDML is given in Appendix C. As shown in Appendix C, SDML captures the derivation results of schema integration, by means of a number of derivation elements, including the following seven specific elements:

- tableRenameDerivation

- tableUnionDerivation
- tableSubtractDerivation
- tableRestrictDerivation
- columnRenameDerivation
- columnUnionDerivation
- columnStringAdditionDerivation.

Everyone of these specific operations, as well as different steps involved for their corresponding schema integration are described later in Section 4.2.5.1. The 'columnStringAdditionDerivation' operation is also used at the end of the schema matching process. This operation enables to define a mapping for specification of the fact that one column in one schema equals to the concatenation of *n* number of columns in the other schema. For example, in order to define a mapping for the matches between the "name" column in the recipient schema and the "first name" and "last name" columns in the donor schema, the 'columnStringAdditionDerivation' operation is applied to "first name" and "last name".

All SDML files generated by SASMINT need to be validated for being compliant with the defined format of SDML. For this purpose, an XML Schema Definition (XSD) is defined in SASMINT for SDML, which is presented in Appendix B. As an example, a part of this XSD, related to *snode* and its derivation constructs is shown in Figure 4.5. Please note that in this example only one type of derivation, the *tableRenameDerivation*, is shown in detail in the figure due to the space considerations. As it can be seen, each derivation consists of one or more (depending on the derivation type) *derivationNode* and zero or one *derivationType* elements. The element named *derivationNode* represents the node(s) from donor and/or recipient schemas participating in the derivation. On the other hand, the element named *derivationType* is a recursive definition and aimed for representing derivations generated at previous integration steps.

4.2.2 Configuration Phase – *P1*

Configuration phase, as represented in Figure 4.3 is the stage for deciding on and assigning proportional weights to each algorithm used for the linguistic and structure matching components of the SASMINT. This phase is also responsible for identifying the selection strategy and setting the threshold regarding the results of the schema matching phase.

Three methods for weight assignment are introduced and supported by SASMINT:

1) Users can choose to apply the SAMPLER component of SASMINT in order to identify the appropriate weights for Linguistic Matching algorithms. The details of this process are provided in Section 4.2.2.1.

2) Users can choose to manually assign weights using their expert advance knowledge about the case, either from past experience with the donor schemas, or their general expertise in information integration.

3) In case neither (1) nor (2) are opted for, SASMINT assumes an equal weight distribution on all applicable algorithms. Needless to say that this might lead to some imprecision and reduce the accuracy of the mapping results. Therefore, in case the user is not experienced, it is always advised by SASMINT approach to apply the SAMPLER component.

```
<xs:element name="snode">
	<xs:complexType mixed="true">
		<xs:choice minOccurs="0">
			<xs:element ref="ns1:tableRenameDerivation"/>
			<xs:element ref="ns1:tableUnionDerivation"/>
			<xs:element ref="ns1:tableSubtractDerivation"/>
			<xs:element ref="ns1:tableRestrictDerivation"/>
			<xs:element ref="ns1:columnRenameDerivation"/>
			<xs:element ref="ns1:columnUnionDerivation"/>
			<xs:element ref="ns1:columnStringAdditionDerivation"/>
		</xs:choice>
		<xs:attribute name="id" type="xs:string" use="required"/>
		<xs:attribute name="name" type="xs:string" use="required"/>
		<xs:attribute name="type" type="xs:string" use="required"/>
		<xs:attribute name="schema" type="xs:string"/>
		<xs:attribute name="table" type="xs:string"/>
		<xs:attribute name="pkColumn" type="xs:string"/>
		<xs:attribute name="refTable" type="xs:string"/>
	</xs:complexType>
</xs:element>
<xs:element name="tableRenameDerivation">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="ns1:derivationNode"/>
			<xs:element ref="ns1:derivationType" minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
<xs:element name="derivationNode">
	<xs:complexType>
		<xs:attribute name="id" type="xs:string" use="required"/>
		<xs:attribute name="name" type="xs:string" use="required"/>
		<xs:attribute name="type" type="xs:string" use="required"/>
		<xs:attribute name="schema" type="xs:string" use="required"/>
		<xs:attribute name="table" type="xs:string"/>
		<xs:attribute name="pkColumn" type="xs:string"/>
		<xs:attribute name="refTable" type="xs:string"/>
	</xs:complexType>
</xs:element>
<xs:element name="derivationType">
	<xs:complexType mixed="true">
		<xs:choice>
			<xs:element ref="ns1:tableRenameDerivation"/>
			<xs:element ref="ns1:tableUnionDerivation"/>
			<xs:element ref="ns1:tableSubtractDerivation"/>
			<xs:element ref="ns1:tableRestrictDerivation"/>
			<xs:element ref="ns1:columnRenameDerivation"/>
			<xs:element ref="ns1:columnUnionDerivation"/>
			<xs:element ref="ns1:columnStringAdditionDerivation"/>
		</xs:choice>
		<xs:attribute name="refDerivationNode" use="required"/>
	</xs:complexType>
</xs:element>
```

Fig. 4.5. Part of XSD for SDML

Furthermore, user can also influence the process for identifying the strategy used for automatic selection of the results of schema matching, as described below:

Method-1. Setting up of a threshold value by user: The user is asked to provide a threshold value for matches, which is used subsequently in this process. Threshold would be defined as a boundary value between 0.0 and 1.0. In similarity calculations, the realized similarity values are compared against this threshold value, and in case they are higher, those pairs are accepted by SASMINT as being similar. If no threshold value is specified by user, a value of 0.5 is defaulted.

Method-2. Getting user's preference (i.e. input) on strategy for selection of match results: The user is provided with the following two strategies to choose from. If nothing is specified by user, the default strategy is select max above threshold.

a. **Select all above threshold** - selecting all matching pairs with similarity above the threshold.

b. **Select max above threshold** - selecting only those with the highest similarity values. If an element of a schema is simultaneously similar to *n* elements of another schema with different similarity values above some threshold, only the highest similarity match is selected for displaying to the user.
However, if the difference between some similarity values of pairs is smaller than 0.01, then it is checked if the parent tables of the pairs match, in order to identify which of the pairs shall be presented to the user as the matched pairs. The algorithm for this strategy is presented in Figure 4.6.
Example: The two cases (Case 1 and Case 2 below) shown in the algorithm of Figure 4.6 are better clarified below through examples, where threshold value is assumed to be 0.5.
Case 1: Suppose that: X is an element of schema S1 and Y and Z are two elements in schema S2. Assuming that similarities of (X,Y) and (X,Z) are computed as 0.6 and 0.7 respectively, SASMINT selects (X,Z) as the matched pair to display to the user.

Similarity values computed by SASMINT:	*Match result displayed to user:*
(X,Y) ➔ 0.6 (X,Z) ➔ 0.7	(X,Z)

Case 2: Suppose that: X, Y, and Z are all columns and X is a column of table T1, Y is a column of table T2, and Z is a column of table T3. Moreover, T1 is a table of schema S1 and T2 and T3 are tables of schema S2. As shown below, suppose that the similarity values computed for both (X,Y) and (X,Z) is 0.7. In order to decide which one of (X,Y) and (X,Z) shall be selected as the matched pair, similarities between the parent table of X (T1) and the parent tables of Y (T2) and Z (T3) are checked. Suppose that similarity of (T1, T2) is above the threshold, but that of (T1, T3) is not. Then the pair whose parent tables match will be selected as the final matched pair to be presented to the user, which is (X,Y) in this example.

Similarity values computed by SASMINT:	*Match result displayed to user:*
(X,Y) ➔ 0.7 (X,Z) ➔ 0.7 (T1,T2) >= threshold (T1,T3)< threshold	(X,Y)

$Given\ T : \{t | Table(t)\}, C : \{c \mid Column(c)\}, S : \{s \mid Schema(s)\}$
$SimS : \{p \mid Pair(p)\}(SimilaritySet), initiallySimS = \varnothing$
$S_1, S_2 \in S$
$Case\ 1 :$
$X, Y, Z \in (T \bigcup C),$
$\exists X, Y, Z \mid X \in S_1, Y \in S_2, Z \in S_2$
$sim(X, Y) = \alpha, sim(X, Z) = \beta$
$if\ ((\alpha - \beta) > 0.01) \Rightarrow SimS = SimS \bigcup \{(X, Y)\}$
$elseif\ ((\beta - \alpha) > 0.01) \Rightarrow SimS = SimS \bigcup \{(X, Z)\}$
$elsegotoCase\ 2$
$Case\ 2 :$
$\exists X, Y, Z, T1, T2, T3 \mid X, T1 \in S_1 and\ Y, Z, T2, T3 \in S_2$
$X, Y, Z \in C$
$T1, T2, T3 \in T$
$T1 = parentTable(X), T2 = parentTable(Y), T3 = parentTable(Z)$
$sim(X, Y) = \beta, sim(X, Z) = \chi$
$if\ (|\beta - \chi| <= 0.01))$
$\{if\ (sim(T1, T2) >= threshold) AND (sim(T1, T3) < threshold) \{$
$SimS = SimS \bigcup \{(X, Y)\}$
$\}$
$elseif\ (sim(T1, T2) < threshold) AND (sim(T1, T3) >= threshold) \{$
$SimS = SimS \bigcup \{(X, Z)\}$
$elseif\ (sim(T1, T2) >= threshold) AND (sim(T1, T3) >= threshold) \{$
$SimS = SimS \bigcup \{(X, Y)\} \bigcup \{(X, Z)\}$
$\}$

Fig. 4.6. Algorithm of *select max above threshold* strategy

4.2.2.1 Sampler Mechanism

SASMINT introduces and implements a composite matching approach. In this approach, the linguistic matching process utilizes a number of algorithms, and combines them by their weighted summation. Linguistic matching algorithms operate on the names of elements. The main reason behind using different algorithms is the complexity and variety of differences between the element names that are compared. Certain algorithms perform better than others in different cases, depending on the element names being matched.

Generating accurate matchings is important in order to reduce the amount of required user input in the process. We consider appropriate distribution of weights as a pre-requisite for achieving accurate matchings. However, deciding on the weights, and assigning them manually by users 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 algorithms that are used in the linguistic matching process. In Figure 4.7,

components involved in the operation of the Sampler Component and their interrelationships are illustrated.

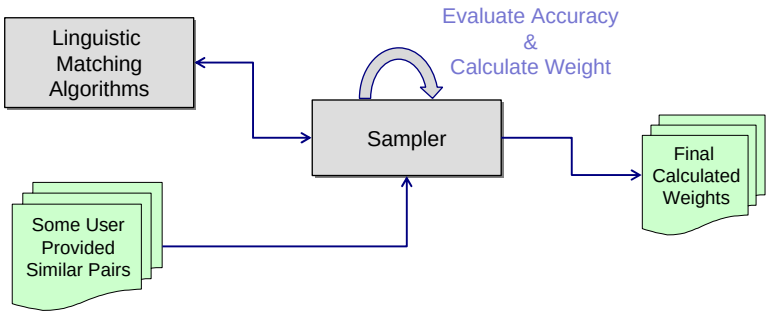


Fig. 4.7 Operation of Sampler

In order to use the Sampler component, users supply a set of similar pairs of element names from their database domain. If the user wishes to have the Sampler calculate suitable weights for syntactic similarity algorithms, then the user needs to provide a set of syntactically similar pairs of names to the Sampler. Similarly, the user needs to provide a set of semantically similar pairs of names for calculating the suitable weights for semantic similarity algorithms.

For a given set of pairs S: {P1, P2, … , PN}, which will be provided by the user and the maximum size of which is suggested to be 5 (N=5) in the current SASMINT implementation, the Sampler runs the syntactic or semantic similarity algorithms for each given pair P in S, and determines their calculated similarity values. The outcome of the calculated similarity for each Pair P is a value between 0 and 1. After the computation of the similarity values, the Sampler starts measuring the accuracy level of each algorithm, using the f-measure (Rijsbergen, 1979) method. F-Measure is a technique used in information retrieval for measuring the quality of a variety of processes, such as the schema matching process. Further details about f-measure are provided in Chapter 6. Using the following formula, the Sampler calculates the suitable weight for each algorithm; where $\sum F$ represents the sum of all f-measure values resulted for all algorithms used, and F_m represents the f-measure value calculated for the algorithm 'm'.

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

As the last step of the weight computation and weight assignment process, the calculated weight of each algorithm suggested by Sampler is provided to the user, through the GUI. At this stage, the user has the option of either accepting the proposed weights, or modifying them as desired.

4.2.3 Automatic Schema Matching Phase – *P2*

Schema matching phase, as represented in Figure 4.3, is the process that aims at identifying all correspondences between the elements of two schemas. This is a crucial component in many

different applications, such as for the general schema integration, for federation of different databases, for providing common access to databases on the Web, etc. Considering the classification of schema matching approaches given in Chapter 2, which is a simplified version of the classification provided in (Rahm & Bernstein, 2001), the SASMINT approach focuses on the schema level matching, while utilizing both the element level information, which corresponds to linguistic characteristics of names of schema elements, as well as the structure level information. Furthermore, SASMINT on one hand exploits a combination of different automatic schema matching techniques for resolving both syntactic and semantic heterogeneity and on the other hand uses the results achieved from schema matching for semi-automatic schema integration. As explained in Section 4.2.2.1, if only a single criterion (for example, name matching) is considered, it is unlikely that achieving high match accuracy for a large variety of schemas will be achieved. As a consequence, it is necessary to combine and utilize multiple techniques at the same time to increase the chances of generating successful results. For this purpose, SASMINT follows a composite matching approach that combines the results of several independently executed match algorithms. This allows for high flexibility, as it creates for all users the potential of applying the best fitting match algorithms to each case based on the specific match task at hand.

Three main activities are involved in the automatic schema matching phase of SASMINT: i) ***preparation***, which translates database schemas into a common Directed Acyclic Graph (DAG) format, ii) ***comparison***, which identifies the correspondences between the two schemas represented as DAGs, resolves their conflicts, and finds out the appropriate matches, using both Linguistic and Structure Matching, iii) ***preliminary result generation***, which displays the results of the schema matching in a graph format. Details of these three main activities are provided in the following sub-sections.

4.2.3.1. Automatic Schema Matching Phase of SASMINT – Preparation Activity

The Preparation activity of schema matching phase deals with the translation of source schemas defined in the typical DDL of their DBMS - the relational DDL - into a common representation format. Searching the literature extensively, we have identified several different alternatives as outstanding candidates for the common representation format, which included the relational data modeling, object-oriented data modeling, UML, XML Schema, Semantic Data Model (SDM) (Hammer & Mcleod, 1981), and Directed Acyclic Graph (DAG). The following criteria have been considered in selecting the common format for representing schemas in SASMINT:

1- Considering the complex nature of semantic interoperability process, the common format to be chosen must be powerful enough to express all different schemas,

2- It must facilitate the automatic matching of schemas, which resolves their syntactic / semantic heterogeneities.

3- It should not be complex for users to understand, as during the Result Generation step, the users are supposed to accept, reject, or modify the suggested mappings by looking at the results in this common format.

Consequently, for SASMINT, the DAG with labeled edges has been chosen as the common format to represent all schemas, considering that it provides a balanced format among other alternatives, supporting the representation of a relational schema, while it can also represent an

object-oriented schema, etc. as a graph. Furthermore, existing graph theory concepts and algorithms can help with comparing two graphs. Additionally, since DAG is not a complex format, users can easily understand schemas represented with these graphs, and therefore it improves the system's understandability by users.

The two schemas that need to be matched in SASMINT are called as the recipient and the donor. User can load the donor schema from a new database system to integrate with others and the recipient schema – the currently integrated schema - either from a different database or from a previously saved XML file. The XML file could have been created at the previous step of incremental schema integration, to capture the integrated schema for federation of several databases. This file is generated in the SDML format, which is introduced in Section 4.2.1. If user chooses to load a schema from a relational database, SASMINT connects to that database and directly downloads the schema related definitions, including tables and columns information from the underlying database's meta-information. During the preparation activity, SASMINT automatically translates the schema definitions into a DAG format. The pseudo code of the preparation activity of SASMINT for loading schemas from a database is shown in Figure 4.8.

```
metadata = getDatabaseMetadata;
tableNameSet = metaData.getTables();
schemaName = metaData.getSchemaName();
schema = generateSchema(schemaName);
graph.addVertex(schema);
while(tableNameSet.hasNext())
  tableName = tableNameSet.next();
  table = generateTable(tableName);
  graph.addVertex(table);
  graph.addEdge(schema, table, hasTable())
  columnNameSet = metaData.getColumns(tableName);
  while(columnNameSet.hasNext())
    columnName = columnNameSet.next();
    column = generateColumn(columnName);
    graph.addVertex(column);
    graph.addEdge(table, column, hasColumn);
  endWhile
endWhile
```

Fig. 4.8. Pseudo Code for Loading Database Schemas

4.2.3.2 Automatic Schema Matching Phase of SASMINT – Comparison Activity

The Comparison activity of schema matching automatically identifies the likely matches between two schemas, using a number of algorithms from NLP and Graph Theory, to resolve their syntactic and semantic as well as their structural heterogeneities. This comparison involves two kinds of matching: Linguistic and Structure, as will be addressed in details in the following sections. The linguistic matching considers only the names of schema elements. On the other hand, the structure matching takes into account the structural aspects of the schemas. In addition to using a combination of algorithms for matching, a heuristic method is also used at this stage for relational schema matching in SASMINT's approach, where the primary key columns of the recipient schema are only compared with primary key columns of the donor schema, and similarly the foreign key columns are only compared with foreign key columns.

Results from linguistic and structure matching are then combined by their weighted summation, in order to determine the similarity of schema elements of the two schemas being compared.

However, before any matching occurs, element names (strings) from the two schemas must be first pre-processed to bring them into a common representation and ready for comparison activity. This is therefore called the pre-processing step and involves the following operations:

1. ***Tokenization and Word Separation:*** By means of tokenization and word separation, strings containing multiple words are split into lists of words, called tokens. For example, the "First Name" is split into "First" and "Name".
2. ***Elimination of stop words:*** Stop words are the common words, such as the prepositions, adjectives, and adverbs that may occur frequently but do not have much effect on the meaning of strings. Hence, they are removed from the names. For example, 'of', 'the', and 'at' prepositions are among the most often used stop words, which will be removed.
3. ***Elimination of special characters and De-hyphenation:*** Similar to stop words, special characters such as '/' and '-' are considered irrelevant for the schema matching process and will be removed from the schema names.
4. ***Abbreviation expansion:*** Since abbreviations are used in schema names extensively, they need to be identified and expanded. For this purpose, a dictionary of well-known abbreviations as well as those specific to the domain for the donor/recipient schemas, is used. For instance, by means of such abbreviation expansion "qty" is expanded to "quantity".
5. ***Normalizing terms to a standard form using Lemmatization:*** Multiple forms of the same word need to be brought into a common base form. Lemmatization is a technique widely used in information retrieval. By means of lemmatization, different forms of the verbs are reduced to the infinitive; also plural nouns are converted to their singular forms, e.g. "knives" is normalized into "knife" and "ate" is normalized into "eat".

4.2.3.2.1 Linguistic Matching

After the element name pairs from two schemas are pre-processed and brought into a common format, their similarity is calculated using a number of matching algorithms from NLP. This process is called Linguistic Matching. The linguistic matching has two main goals, namely identifying both the *syntactic* and the *semantic* similarity between pairs of element names. A combination of string similarity algorithms are utilized to determine syntactic similarity, while semantic similarity algorithms in SASMINT use the WordNet, which is a lexical database, and considers a variety of relationships between the terms, such as synonymy (e.g. "price" is a synonym of "cost"), hypernymy (e.g. "color" is an hypernym of "blue"), hyponymy (e.g. "blue" is a hyponym of "color"), holonymy (e.g. "hand" is a holonym of "finger"), and meronymy (e.g. "finger" is a meronym of "hand"). Linguistic matching algorithms are typically called as *measure* or *metric*. Both measure and metric have the same meaning when these algorithms are considered, and thus they are used interchangeably.

The result of the linguistic matching is the similarity value between each considered pairs, which is in the range of [0,1], where the value 1 indicates the full equality between the terms compared. For linguistic matching, the syntactic and semantic similarity results are combined in the following manner: First, the syntactic similarity of a pair of element names is identified.

If this similarity is above the similarity threshold, which is set at the configuration phase as described in 4.2.2, then the semantic similarity is not checked, and the result of the linguistic matching value for this pair becomes the calculated syntactic similarity value. Otherwise, the semantic similarity of the pair is determined. Again, if this result is above the threshold, then the result of linguistic similarity of this pair becomes the semantic similarity value. However, if neither syntactic and nor semantic similarity values of the pair are above the threshold, then the linguistic matching value is the average of the two resulted values. A pseudo code of the Linguistic Matching is given in Figure 4.9. Further details of syntactic and semantic similarity are provided in the following subsections.

```
Inputs: S1 in 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_S1 X 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
    endElse
    if (sem > threshold)
       LinguisticMatch = sem
    endIf
    else
       LinguisticMatch = weight(syntactic) * syn + weight(semantic)*sem
    endElse
endFor
```

Fig. 4.9. Pseudo Code for Linguistic Matching

I. *Syntactic Similarity*

There is a large number of well known algorithms coming from the natural language processing community, which try to identify the syntactic similarity values. As stated in the previous section, these algorithms are usually called metrics or measures.

Syntactic similarity metrics are classified as *string-based*, *token-based*, and *hybrid* (Cohen et al., 2003). *String-based* similarity metrics consider strings as streams of characters and do not divide multi-word strings into substrings. *Token-based* similarity metrics view strings as unordered sets of tokens. *Hybrid* similarity metrics combine string-based and token-based similarity metrics such that strings are split into tokens, but then a string-based metric is applied to each token. For example, consider two strings "student number" and "number of students". A string-based metric would operate on these two given strings as they are stated, while a token-based similarity metric would first perform the decomposition of these two strings into their tokens, i.e. {"student", "number"} for the first string and {"number", "of",

"students"} for the second string and then treat each token as a separate string when applying the formula (such as the formula of Jaccard) for computing the similarity.

Another dimension considered for classification of metrics deals with how the results from running algorithms are represented. One type of metrics, called *similarity metrics*, results in a value between zero and one, i.e. [0,1], where higher values indicate closer similarity. On the other hand, the result generated by another type of metrics, called *distance metrics*, is an integer number bigger than or equal to zero, where higher values indicate less similarity between the strings that are being compared.

As addressed in Section 4.1.2, previous schema matching approaches typically depend on only one metric. However, considering that each of these existing metrics is in practice best suited for a different class (i.e. type) of strings, this approach for schema matching is not effective. Namely, for some types of element names, some similarity metrics do not perform well, while another metric performs adequately. Aiming to overcome the limitations of utilizing only a single metric, as a part of the SASMINT approach, a weighted sum of a combination of several mainstream syntactic similarity metrics is used to syntactically compare every two character strings introduced in schemas, thus making it a more generic automated tool to be used for different types of strings. Each of the metrics considered in SASMINT is briefly explained below. In order to help the reader better understand these metrics, a glossary of the terms and symbols used by syntactic similarity metrics is provided in Table 4.6.

Table 4.6- A Glossary of Terms and Symbols Used by Syntactic Similarity Metrics

Term/Symbol	Definition
String distance	Measure of how dissimilar two strings are. Higher value indicates less similarity
String similarity	Measure of how similar two strings are. Higher value indicates more similarity
Operation	Inserting, deleting, or substituting one or more characters
Modification	Act of changing a string by removing or introducing characters
Cost	Measuring operation complexity, by application of a unit of algorithmic complexity (e.g. one letter modification may cost 1)
$\|x\|$	Number of characters in a given string, i.e. the length of a string
Affine gap model	A measure that is used in sequence alignment and encourages the extension of gaps rather than the introduction of new gaps
Character	An alphanumeric symbol which is the smallest indivisible entity of a string.
$x \bigcup y$	Union of strings x and y, i.e. union of words in strings x and y
$x \bigcap y$	Intersection of strings x and y, i.e. common words of strings x and y

1. *Levenshtein Distance (Edit Distance)* (Levenshtein, 1966), also known as Edit Distance, is based on the idea of minimum number of modifications required to change one string into another. Each modification has a cost of 1. Levenshtein distance is a string-based distance metric. Since the result of the syntactic similarity process in our approach is a value between [0,1] the distance value obtained by the

Edit Distance metric is converted to a similarity value in this range using the following formula that we have introduced:

$$sim(A,B) = 1 - (\frac{LD(A,B)}{\max(|A|,|B|)})$$

Levenshtein distance is suitable for common typing mistakes, but not suitable for some cases. For example, when this metric is used, "Blue Apartment" and "Blue Apt." are found as less similar than "Blue Apartment" and "Bold Apartment".

2. *Monge-Elkan Distance* (Monge & Elkan, 1996) is another string-based distance function using an "affine gap model". Affine gap model takes its roots from the sequence alignment, which is used to identify the similar regions in DNA, RNA, and protein sequences. Affine gap model is based on two types of costs: one for opening the gap and another for extending the gap. This model encourages the extension of gaps rather than the introduction of new gaps. The cost of a gap is computed as $\cos t(g) = a + b*l$ where a is the cost of opening a gap, b is the cost of extending a gap, and l is the length of a gap. Monge Elkan Distance works better than the Levenshtein Distance for the shortened strings, such as "Ilker Murat Karakas" vs. "Ilker M. Karakas". It allows sequences of mismatched characters.

3. *Jaro* (Jaro, 1995), a string-based metric, well known in the record linkage community, is intended for short strings. This metric takes into account insertions, deletions, and transpositions as well as the spelling variations, such as "Isabella" and "Isabel". Given two strings A and B, Jaro similarity metric is calculated as follows:

$$Jaro(A,B) = 1/3 * (\frac{CT_A}{|A|} + \frac{CT_B}{|B|} + \frac{CT_A - T_{A,B}}{CT_A})$$

where CT_A is the number of terms in A that match some terms in B, and CT_B is the number of terms in B that match some terms in A. Two terms a_i in A and b_j in B are considered matching if $i - H \le j \le i + H$ where $H = \frac{\min(/A/,/B/)}{2}$. For example, consider the two strings "credit" and "reditc". Although other characters match, these two "c"s do not match, because "c"'s position in the second string needs to be somewhere in 1-3< j< 1+3, but its position is 6. The $T_{A,B}$ in the formula of Jaro is half the number of transposed characters for both strings. Transposed characters are the ones that are matching, but in different order in the two strings. For example, in SUIT and SUTI, I and T are the transposed characters.

4. *TF*IDF (Term Frequency*Inverse Document Frequency)*(Salton & Yang, 1973) is a vector-based approach from the information retrieval research. Weights are assigned to terms in respect to their frequency within the predefined corpus, (typically the internet) and the inverse frequency within the test string or document. 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 (Cohen et al., 2003).

5. *Jaccard Similarity* (Jaccard, 1912) is a token-based similarity measure yielding a similarity value between 0 and 1. The Jaccard similarity between two strings A and B consisting of one or more words is defined as the ratio of the number of shared words of A and B to the number of words contained in A or B. In other words, it is defined as the size of the intersection divided by the size of the union of the two strings. For example, suppose that string A is "student_grade" and B is "student_phone". Then, the number of shared words of A and B is 1 ("student") and the number of words owned by A or B is 3 ("student", "phone", and "grade", where "student" is counted only one time). The formula for the Jaccard similarity is as follows:

$$Jaccard(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

where $|A \cap B|$ is the number of words in $A \cap B$ and $|A \cup B|$ is the number of words in $A \cup B$.
The Jaccard is suitable for comparing long strings consisting of a number of words, such as addresses. Since it compares the words in strings, it is word order independent. Especially in shorter strings, this metric is sensitive to misspelled terms. For example, while Jaccard measure finds 100% match between the strings "Ozgul Unal 2. Street 06560 Ankara" and "Unal Ozgul 2. Street Ankara 06560", it performs badly for strings "Ozgul Unal" and "Unal Ozgur".

6. *Longest Common Subsequence (LCS)* is a special case of edit distance. The longest common subsequence of A and B is the longest run of characters that appear in order inside both A and B. Both A and B may have other extraneous characters along the way, and thus LCS is suitable for the cases where strings might have spelling errors. For example, the LCS of two strings "ACGGA" and "CGAG" is "CGA". Different from the edit distance, if the value of LCS is higher, strings are more similar. We use the length of the LCS to identify the similarity of scores using the following formula:

$$LCS(A,B) = \frac{|LCS|}{\min(|A|,|B|)}$$

where $|LCS|$ is the length of the LCS of two strings compared, which are A an B.

Syntactic Similarity Metrics Used in SASMINT

SASMINT uses all six metrics described above in order to effectively and automatically identify the syntactic similarity between every two schema element names. Considering that on one hand each metric is in fact most suitable for certain specific type of strings (e.g. Jaro is intended for short strings, Monge-Elkan distance allows sequence of mismatched characters, etc.), and that on the other hand schemas usually consist of mixed sets of element names (strings), the approach of SASMINT benefits from applying a combination of these metrics and therefore obtains more accurate results. At the high level, these metrics are combined by their weighted summation, using the following formula:

$$sim_{WSyntactic}(a,b) = w_{lv} * sm_{lv}(a,b) + w_{me} * sm_{me}(a,b) + w_{jr} * sm_{jr}(a,b) + w_{tf} * sm_{tf}(a,b) + w_{jc} * sm_{jc}(a,b) + w_{lc} * sm_{lc}(a,b)$$

where ‘sm’ is the similarity score, ‘w’ is the weight, ‘lv’ stands for Levenshtein, ‘me’ for Monge-Elkan, ‘jr’ for Jaro, ‘jc’ for Jaccard, ‘tf’ for TF-IDF, and ‘lc’ for Longest Common Subsequnce. As explained in section 4.2.2, the weight (‘w’) for each metric is identified and set at the Configuration phase. After being set, the weights are not changed throughout the schema matching and schema integration phase.

II. *Semantic Similarity*

Identifying the semantic similarity between two words or concepts has been the subject of many applications in Natural Language Processing (NLP), information retrieval, and federated databases among other areas. Another term that is frequently used together with semantic similarity is the semantic relatedness. If two concepts are related using any kind of relation, then that means they are semantically related. As (Budanitsky & Hirst, 2001) emphasizes, semantic relatedness is more general than semantic similarity and covers a broader range of relationships, while semantic similarity is mostly limited to IS-A relations. A number of semantic similarity and semantic relatedness algorithms that are widely used in NLP research domain are described below.

Typically, the semantic similarity measures utilize a variety of knowledge resources, e.g. Roget’s Thesaurus (Kirkpatrick, 1998) and WordNet (Fellbaum, 1998). Most measures in fact utilize WordNet.

The WordNet is a dictionary of nouns, verbs, adjectives, and adverbs, which are organized into synonym sets, each representing one underlying lexical concept. Synonym sets, also called as synset, are interlinked by different relations, such as hypernymy, hyponymy, antonymy, meronymy, holonymy, etc. Since in our application we deal with schema element names, which are mainly nouns, and hypernymy / hyponymy (representing IS-A) is the most dominant relationship linking nouns in schemas, we only apply this relationship and the synonymy, as introduced in the WordNet, when identifying semantic similarity of element names. A concept X is hyponym of a concept Y, if X ‘is a kind of’ Y. On the other hand, hypernym represents a more general entity than the hyponym. For example, “art student” is a hyponym (e.g. subclass) of “student”, whereas “person” is a hypernym (e.g. superclass) of “student”. In order to find the hyponyms of a concept, one needs to go down in the WordNet IS-A hierarchy.

Partially inspired by the approach of (Pedersen et al., 2005), we categorize semantic similarity and semantic relatedness measures into three groups: a) *path-based* measures, b) *information content* measures, and c) *gloss-based* measures, which are discussed at length below.

a) ***Path-based Measures:*** The main idea behind these measures is calculating the shortest path between the concepts in the IS-A hierarchy, such as the IS-A hierarchy of WordNet. As an example, the measure introduced by Leacock and Chodorow (Leacock & Chodorow., 1998) is based on the length of paths between noun concepts in an IS-A hierarchy. They compute the shortest number of links from one node in WordNet to another, using breadth-first search. Another semantic similarity measure, which also uses path length, is that of Wu and Palmer (Wu & Palmer, 1994). They focus however on verbs and take into account the lowest common subsume of the concepts. Hirst and St-Onge (Hirst & St-Onge, 1998) extend the path length measure to include all relations in WordNet and penalizing the changes in direction. Also, since it is not restricted to IS-A relations, it is called as semantic relatedness measure. It clusters the relations in WordNet

in three ways, namely as horizontal, up, and down. Hirst and St-Onge also define four levels of relatedness: extra strong, strong, medium strong, and weak. Within the scope of SASMINT, we only focus on measures based on IS-A hierarchies, and therefore do not apply the details of the Hirst and St-Onge measure.

b) ***Information Content Measures*:** Using only the path length may cause some problems. For example, in the lower part of the WordNet hierarchy, terms have more similarity and thus links between them represents a shorter semantic distance than links between the terms near the root, where terms are less similar. Path length cannot differentiate between these two cases. For example, semantic distance of "cat" and "tiger" is shorter than that of "animal" and "organism", but path length may identify these pairs as equally similar. In order to deal with these problems, other types of measures have been proposed, which exploit the Information Content. These measures typically employ text corpus statistics about the concepts, in order to assign the information content value to them. In order to compute the information content value, measures follow the formulation of information theory, where information content IC for any concept c is defined as:

$$IC(c) = -\log p(c)$$

In this formula, p(c) is the probability of encountering an instance of concept c. The probability of concepts that are higher in the hierarchy will also be higher, as the frequency of a concept includes the frequencies of its subordinates. However, higher probability means lower information content, so the concepts appearing higher in the hierarchy are less informative than the ones appearing at lower levels.
The measure of Resnik (Resnik, 1995) is an example of information content measure, which uses hyponymy relation. The information content values are derived from the word frequencies in the Brown Corpus. The frequency of a word is calculated by counting the number of occurrences of the word type in a corpus, and dividing that count by the number of different concepts / senses associated with that word. The semantic similarity between two concepts is then proportional to the amount of information that they have in common and defined as follows:

$$sim_{res}(c_1, c_2) = IC(lcs(c_1, c_2))$$

where sim_{res} is the Resnik semantic similarity and *lcs* is the lowest common subsumer (also known as maximally specific superclass) of concepts .
One limitation of the measure of Resnik is that a large number of concepts might have the same least common subsumer, and thus have identical values of similarity.
Another measure, using information content of nodes in a IS-A hierarchy is proposed by Jiang and Conrath (Jiang & Conrath, 1997). They consider the information content of the concepts themselves along with the information content of their lowest common subsumer. Instead of semantic similarity, they calculate semantic distance $dist_{jc}$ of two concepts, c_1, c_2, as follows:

$$dist_{jc}(c_1, c_2) = IC(c_1) + IC(c_2) - 2 * IC(lcs(c_1, c_2)$$

c) ***Gloss-based Measures:*** These types of measures utilize gloss overlaps. Gloss refers to a brief description of a word. For example, the gloss provided by WordNet for one sense of the word "building" is "a structure that has a roof and walls and stands more or less

permanently in one place". One disadvantage of this kind of measure might be that since glosses are short, they may not provide adequate information.

Lesk (Lesk, 1986) uses gloss overlaps for word sense disambiguation. Lesk counts the number of common words between the glosses of each sense of a target word, and glosses of other words in a sentence.

Banerjee and Pedersen (Banerjee & Pedersen, 2002) modify the Lesk algorithm such that in addition to computing the overlaps between the glosses of the senses of two concepts, they also consider the glosses of the senses of the concepts that are semantically or lexically related to the two concepts.

SASMINT utilizes the two measures of the Path-based and Gloss-based. The information content measures are not utilized, because the choice of corpus (i.e. information content source) might have an unpredictable impact on the results that one gets from using these types of measures. The performance of information content measures is influenced by the corpus used (Patwardhan, 2003). Furthermore, the fact that one cannot foresee how the selection of a particular corpus would affect the matching performance makes the selection of the corpus even more difficult. Among several introduced alternative approaches for the path-based and gloss-based measures, we have chosen two that are widely known measures. These are explained below:

1. *Path-based Measure:* SASMINT utilizes the measure introduced by Wu and Palmer (Wu & Palmer, 1994) as the path-based measure. Wu and Palmer calculate the semantic similarity of two concepts, using the following formula:

$$sim_{wup}(c_1, c_2) = \frac{2 * N_3}{N_1 + N_2 + 2 * N_3}$$

where sim_{wup} is the Wu and Palmer semantic similarity, N_3 is the number of nodes on the path from root to the maximally specific superclass c_3 of the c_1 and c_2. N_1 is the number of nodes on the path from c_1 to c_3, and N_2 is the number of nodes on the path from c_2 to c_3.

Resnik has modified this formula slightly, resulting in the following formula (Resnik, 1999):

$$sim_{wup}(c_1, c_2) = \frac{2 * depth(lcs(c_1, c_2))}{depth(c_1) + depth(c_2)}$$

where sim_{wup} is the modified Wu and Palmer semantic similarity, *depth* is the distance from the root node and $lcs(c_1, c_2)$ is the maximally specific superclass of c_1 and c_2.

2. *Gloss-based Measure:* The other type of measure used in SASMINT for determining semantic similarity is based on the gloss overlaps. We get the gloss information from WordNet. The measure of Lesk (Lesk, 1986) forms the base for the gloss-based measure used in SASMINT. A word can have different senses, depending on the context. In SASMINT, we customize the algorithm of Lesk to compute the semantic similarity of two concepts c_1 and c_2 as follows: for each of the senses of c_1, we compute the number of common words between its glosses and the glosses of each of the senses of c_2.

Semantic Similarity Metrics Used in SASMINT

Similar to the case in the approach for syntactic similarity of SASMINT, the approach for semantic similarity also considers the combined weighted sum of two semantic similarity measures, as addressed above. Following formula is used for computing the final result of semantic similarity in SASMINT:

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

where 'wup' stands for Wu and Palmer's measure and 'gloss' stands for the gloss-based measure, w_{wup} is the weight for Wu and Palmer's measure, $sm_{wup}(a,b)$ is the similarity value calculated by using the Wu and Palmer's measure, w_{gloss} is the weight for the gloss-based measure, and $sm_{gloss}(a,b)$ is the similarity value calculated by using the gloss-based measure.

By default, the weight is equally distributed over these two measures. Alternatively, the sampler component of SASMINT can be used to determine the appropriate weights. SASMINT uses WordNet as the base to identify the path between the concepts being compared. Similarly, it benefits from the gloss information provided in the WordNet for calculating its Gloss-based similarity.

4.2.3.2.2 Structure Matching

In addition to linguistic differences, other types of differences are also frequently observed among database schema definitions related to the structures defined among schema elements. Structural differences are more difficult to resolve than linguistic differences and they can be only semi-automated, typically requiring the user's involvement and input. Therefore, the second step of schema matching in SASMINT is focused on the structure matching. As explained in Section 4.2.3.1, before the comparison activity starts, two schemas are represented as graphs. Structure matching takes as input these two graph representations and uses the results generated by linguistic matching step, in order to as much as possible identify the structural similarity between these two schemas. SASMINT's approach for structure matching of schemas is mostly based on the idea that if two elements have been found to be similar, then their adjacent elements in the schemas (parent and children nodes) may also match. Moreover, similarity of two nodes is directly affected by the number and quality of the similarity among their children.

For the purpose of structure matching, a variety of graph similarity and graph matching algorithms from the Graph Theory as well as the web searching, and the schema matching were considered. A number of different notions are introduced for similarity in graphs, as stated in (Zager, 2005), each addressing certain specific questions, including: Are the two graphs identical copies of each other? How much change is needed to convert one graph into the other? Do they contain a common subgraph? Aiming to answer these questions has resulted in the introduction of different types of similarity notions in graphs, such as the graph isomorphism, maximum common subgraphs, minimum common supergraphs, and error tolerant matchings.

Graph isomorphism shows that graphs are structurally equivalent. The complexity of isomorphism algorithms is still vague but it is thought to lie between the P- and NP-complete complexity classes (Foggia et al., 2001; Zager, 2005). The "subgraph isomorphism" is the

generalization of the graph isomorphism, where isomorphic copies of a graph are searched within another graph.

Other definitions related to graph similarity are maximum common subgraph and minimum common supergraph, which are generalizations of the subgraph isomorphism. The maximum common subgraph of two graphs G1 and G2 is the largest graph contained in both G1 and G2. The minimum common supergraph of two graphs G1 and G2 is the smallest graph that contains both G1 and G2 (Bunke, 2000).

Another notion in graph similarity is error tolerant matching using graph edit distance (Bunke, 2000), which is the extension of string edit distance. Edit operations of type insertion, deletion, and substitution can be applied to the nodes and edges of graphs. Graph edit distance measures the minimum number of edit operations required to transform one graph into another.

In graph similarity research field, there are several iterative algorithms introduced, which are based on the mere idea that two nodes of two graphs are similar if the neighbors of these nodes are also similar. One such iterative algorithm is that of Kleinberg, which is named as Hub and Authority Scoring (Kleinberg, 1999). The algorithm is motivated by the fact that the information content of a web page is not only the sum of the information in the page itself, but also includes the other pages linked to or being linked by this page. Kleinberg's algorithm identifies in a set of pages, relevant to a query search, the subset of pages that are good hubs or good authorities. Then, the result of the query return both the authorities, which contain the primary content related to the query, and the hubs which points at sources containing primary content related to the query (authorities). The iterative method assigns an authority score and a hub score to every vertex of a given graph. The hub score of a vertex is equal to the sum of the authority scores of all vertices pointed to by the vertex itself. The authority score of a vertex is equal to the sum of the hub scores of all vertices pointing to the vertex. Given that B is the adjacency matrix of a graph G, and that h and a are the vectors of hub and authority score, the iterative method is defined as follows:

$$\begin{bmatrix} h \\ a \end{bmatrix}_{k+1} = \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix} \begin{bmatrix} h \\ a \end{bmatrix}_k$$

A generalization of the Kleinberg's algorithm that computes the similarity of two graphs G_A and G_B with the vertices n_A and n_B and edges E_A and E_B is proposed in (Blondel et al., 2004). For $i = 1,..,n_B$ and $j = 1,..,n_A$ the similarity scores are updated iteratively using the following equation:

$$X_{k+1} = BX_kA^T + B^TX_kA$$

where X_k is the n_B x n_A matrix of entries x_{ij} at iteration k, A and B are the adjacency matrices of G_A and G_B, and A^T and B^T are the transpose of A and B. Then, based on this equation, (Blondel et al., 2004) define the following equation to iteratively compute the similarity matrices of graphs:

$$Z_{k+1} = \frac{BZ_kA^T + B^TZ_kA}{\left\| BZ_kA^T + B^TZ_kA \right\|_F} \quad \text{k=0,1,….}$$

where Z_k is the similarity matrix at iteration *k*. The matrix norm, which is $\|\cdot\|_F$, used here, is known as the Euclidean or Frobenius norm and equals to the square root of the sum of all squared entries. The matrix subsequences Z_{2k} and Z_{2k+1} converge to Z_{even} and Z_{odd}. Iteration continues an even number of times and stops upon convergence.

In addition to graph similarity and matching algorithms in Graph Theory domain, a number of other algorithms are also proposed for the schema matching domain, such as the structure matching algorithms of Cupid (Madhavan et al., 2001) and the Similarity Flooding (Melnik et al., 2002). Structure matching by Similarity Flooding (Melnik et al., 2002) is based on a fix point computation. It is based on the assumption that whenever any two elements are found to be similar, similarity of their adjacent elements increases. Over a number of iterations, the initial similarity of any two nodes propagates through graphs. The algorithm terminates after the similarities of all model elements stabilize.

After examining the above mentioned types of graph similarity and structure matching algorithms, two approaches described above were identified as most relevant and applicable for the specific case of schema structure matching and therefore SASMINT has adapted and applied the graph similarity algorithm proposed in (Blondel et al., 2004) and the structure matching by Similarity Flooding proposed in (Melnik et al., 2002).

Structure Similarity Algorithms used in SASMINT

The graph similarity algorithm of (Blondel et al., 2004) and the structure matching algorithm of Similarity Flooding (Melnik et al., 2002) together form the base for the structure matching in the schema matching phase of SASMINT. Similar to the method followed in linguistic matching, structure matching uses the combined weighted sum of these two structural similarity algorithms, as shown in the formula below:

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

where 'blondel' stands for the algorithm of (Blondel et al., 2004) and 'sf' stands for the algorithm of Similarity Flooding.

Since it is not possible to automatically identify the weights of structure similarity algorithms by only providing the element-name pairs, unlike for the linguistic matching, the SASMINT's Sampler component cannot be applied to the structure matching algorithms. Furthermore, success of structure matching also depends on the accuracy of the results of linguistic matching. Therefore, either the weight for each of the structure matching algorithms is defined by the user, or the SASMINT approach assumes equal weight distribution for the above two algorithms as the default.

SASMINT aims to resolve a number of structural conflicts, as addressed in Section 3.3. While in most cases, SASMINT is able to fully automize this process, in some cases the process is semi-automated, since user input is required to resolve some structural conflicts. Consider a simple example, related to the case of attribute-attribute conflict: Suppose that *name* information is stored in the "name" column of the first schema and in the "first_name" and "last_name" columns of the second schema. Although SASMINT can identify the match between (name - first_name) and (name - last_name), users need to then specify through the GUI of SASMINT that "name" is equal to the concatenation of the "first_name" and "last_name".

As such, in general, fully automatic resolution is not realistic to be expected for all types of semantic and structural conflicts. Therefore, although some user input might be required in some cases, SASMINT addresses and handles all the conflicts addressed in Section 3.3.

4.2.3.3 Automatic Schema Matching Phase of SASMINT – Preliminary Result Generation Activity

In the SASMINT approach, the results of the linguistic and structure matching are combined in order to generate the final similarity values between each element name pairs. Namely, in SASMINT, final similarity is calculated using the following formula:

$$sim_{Final}(a,b) = w_{Linguistic} * sim_{WLinguistic}(a,b) + w_{Structure} * sim_{WStructure}(a,b)$$

where $sim_{WLinguistic}$ is the result of linguistic matching. It is computed based on the $sim_{WSyntactic}$ and $sim_{WSemantic}$ values, applying the algorithm, presented in Figure 4.9. In the above formula, $sim_{WStructure}$ represents the result of structure matching, $w_{Linguistic}$ is the weight of linguistic matching, and $w_{Structure}$ is the weight of structure matching. Since structure matching uses the results of linguistic matching as the base and linguistic properties have higher effect on the similarity of schema elements, the linguistic matching also has higher influence on the final similarity calculation results. Therefore, the weight of linguistic matching in SASMINT formula is currently defaulted as 0.70, while the weight of structure matching is set to 0.30, in the implementation of SASMINT. These weights are selected since they proved to be appropriate for all the experiments that we have performed in this research work (see sections 6.5 and Appendix E), nevertheless they are modifiable by user through the GUI if needed to better fit other potential cases.

The Comparison activity results in similarity values between all element name pairs. Based on the threshold value and the selection strategy defined in the configuration phase, similar pairs are identified. Results consisting of these similar pairs are displayed to users in two formats: 1) graph format, with edges between the tables and their columns as well as between the nodes identified as similar. 2) text format showing the results of each algorithm used in the comparison activity. This format is necessary in order to guide users for the future match processes, to decide on what weight to assign to which algorithm for what kind of schema elements as well as for enabling clear understanding of the results.

4.2.4 User Modification and Validation Phase – *P3*

The fact that the Schema Matching and Integration are activities, which do not lend themselves to a fully automated set of computational activities (i.e. requiring no user involvement), there is always the need to support a human in the loop. This is especially the case considering the typical existence of a large amount of implicit semantics involved in schema descriptions, which may be discovered or assumed by human intelligence. Therefore, we identify this phase of the SASMINT approach; the human-in-the-loop phase, which takes place after the Automatic Schema Matching Phase. This phase is called the '*User Modification and Validation*' phase in the SASMINT system. Without such a phase, it would not be possible to assure the identification of all the matches between the two schemas. In order to facilitate the user interaction, a GUI plays an important role in this phase.

From a process point of view, the logical order of activities that take place in this phase are recapped as follows:

a) The visualization of the candidate matching results to the user by means of a GUI.

b) Application of user's modifications on the match results. Here, the set of possible modifications comprise:

   1) Introduction of some new match relationships by the user

   2) Removal of some computer proposed match relationships

   3) Modification of some computer-proposed match types

c) Capturing and persistence of the match results

An example case, for which the user input is essential, occurs in all complex cases, such as for *1-to-n* matches (one column in one schema matches more than one column in the other schema). For this case, it is not possible to only automatically decide whether a column in the first schema is a combination of *n* columns in the second schema and even if so, it may not be known how to combine these *n* columns, e.g. through using: concatenation, summation, etc. As a simple example for clarifying this case, suppose that schema matching system has identified a match between the "address" element in one schema and two other elements, namely "addr" and "dress" in the second schema. In this case, user is supposed to delete the match between "address" and "dress" as it is a meaningless match, and perhaps validate the other match. As another example, suppose that the system has identified a match between the "address" element in one schema and "street", "zip", and "city" elements in the second schema. In this case, user is supposed to further select/specify that "address" is the *concatenation* of "street", "zip", and "city".

4.2.5 Schema Integration Phase – *P4*

Schema integration phase, as represented in Figure 4.3, is a key process in different database applications. Schema integration is a necessary step for database interoperability, federation, and supporting the ultimate co-working among different nodes in the network. Nevertheless, it is a difficult process because of the many structural and linguistic conflicts among schemas, and performing it manually for all nodes in the network is very time consuming, cumbersome, and error prone. Consequently, it is highly desirable to assist the users through semi-automation of this process. Therefore, in SASMINT introducing novel approaches are aimed to automate the schema integration using the results generated through the schema matching phase. For instance, in federated database systems, in order to generate a federated schema, the schema that is local at a participating node needs to be integrated with the parts of schemas that other nodes share with this node. As another example, and following the global schema approach, schemas of all nodes in a collaborative network need to be integrated together, with the aim of generating a single global schema for the network. SASMINT facilitates the schema integration process by providing supervised automated means to achieve schema integration. As such, for every two schemas, after saving the results of their validated schema matching results, the option exists for the user to continue with generating their integrated schema.

Two important components of schema integration in SASMINT are the derivation constructs and the integration rules. Derivation constructs are used to keep the derivation history for integration purposes, as explained in details later in Section 4.2.5.1. Considering that a number of different integration conflicts need to be resolved for reaching a successful integration of schemas, a number of rules are defined in SASMINT to be used for automatic integration of relational schemas. Explanations at length related to each of these rules are provided later in Section 4.2.5.2. These integration rules operate on different types of match results, to generate their automatic integration. For example, these rules identify which tables and columns need to

be inserted in the resulting schema and with which structure they need to be merged etc. in order for the integrated schema to represent all elements of the two participating schemas.

Using the derivation constructs and the integration rules, introduced in SASMINT, the schema integration operates as follows: First, the schema integration rules are applied in the order given in Section 4.2.5.2. Whenever a rule is applicable, one or more derivation constructs are used to automatically generate the derivation of integration results. In other words, for each newly generated table and column in the integrated schema, the derivation constructs in the SDML format formally specifies where this new element comes from (all source nodes) and how it is generated from its source nodes. When the recipient and donor schemas are integrated, both the elements of this integrated schema as well as the derivation information for each of these elements are displayed to the user.

In order to make the process of schema integration more clear, as a very simple example, suppose that there are following two schemas that need to be integrated. These schemas are called as the recipient and the donor respectively. Column and table names in recipient schemas are the ones that shall remain in the integrated schema.

Recipient Schema: *apartment (no, address)*

Donor Schema: *building (number, addr, floor)*

After the schema matching phase of SASMINT, the identified and validated similar pairs are as follow: *(apartment, building), (no, number)*, and *(address, addr)*. The schema integration process operates on these results of the schema matching as follows:

1) Rule 3 (Section 4.2.5.2) is applied for *(apartment, building)* match;
    a. A new table in the integrated schema is generated using the name of the table in the recipient schema: *apartment*
    b. Non matching columns of recipient and donor tables are added to this new table: *apartment (floor)*
    c. *Table Union* derivation rule is applied to define that the table *apartment* in the integrated schema is the union of the two tables, *apartment* and *building* from the recipient and donor schemas respectively.
    d. *Column Rename* derivation rule is applied to define that the *floor* column of the *apartment* table in the integrated schema is the renamed version of the corresponding column of the *building* table.
2) Rule 13 is applied to the match pair *(no,number)*, therefore:
    a. A new column, named *no* is generated as a column of the new *apartment* table in the integrated schema.
    b. *Column Union* derivation rule is applied to define that this new column is the union of the *no* and *number* columns from the *apartment* and *building* tables (as the recipient and donor schemas) respectively.
3) Rule 13 is applied for *(address,addr)* match:
    a. A new column named *address* is generated as a column of the new *apartment* table in the integrated schema.

b. *Column Union* derivation rule is applied to define that this new column is the union of the *address* and *addr* columns from the *apartment* and *building* tables (as the recipient and donor schemas) respectively.

The final integrated schema would be as follows: *apartment (no, address, floor)*. The integrated schema together with the specification of the schema elements, and the derivation information will be represented and stored in the SDML format.

Below, as a first step, subsection 4.2.5.1 defines the SASMINT's derivation constructs and provides examples for each of them. Then, as a second step, subsection 4.2.5.2 provides the details of the SASMINT's schema integration rules.

4.2.5.1 Schema Integration Phase of SASMINT - Derivation Constructs

The Schema Integration phase of SASMINT introduces the usage of a number of derivation constructs for representing the integrated schemas. Derivation constructs enable definition of how and from which elements of recipient and/or donor schemas, the elements of integrated schema are generated. The definition of these constructs are rooted in and presents a variation of the PEER derivation language (Afsarmanesh et al., 1994). Two main types of derivation constructs are defined for relational schemas: 1) ***Table Derivations*** - consisting of the derivation constructs related to "Table Rename", "Table Union", "Table Subtract", and "Table Restrict"; 2) ***Column Derivations*** - comprising of derivation constructs related to "Column Rename", "Column Union", and "Column Extraction". Some of these derivation operations, namely: Table Rename, Table Union, Column Rename, Column Union, and Column Extraction are those frequently used by the SASMINT's automated schema integration approach. Formal representation of the above mentioned derivation constructs is given below:

1) Table Derivation: A Derived Table is defined by the following expression:

derived –table-definition :=derived-table-name = <T-expr>

T-list:= <T-expr> | <T-expr> , <T-list>

T-expr:= table-name@schema-name | union (<T-expr> , <T-list>) |

subtract (<T-expr> , <T-expr>) | restrict (<T-expr> , <restriction>)

Table derivation primates, used in the expression above are defined further below, where every T_i stands for *table-name@schema-name* and *T* represents the *derived table*:

1. *Table Rename*

   $T = T_1$

2. *Table Union*

   $T = union(T_1,..,T_n)$

3. *Table Subtract*

   $T = subtract(T_1,T_2)$

4. *Table Restrict*

   $T = restrict(T_1, restriction)$

2) Column Derivation: A Derived Column is defined by the following expression:

derived–column-definition :=derived-column-name = <c-expr>

c-list:= <c-expr> | <c-expr> , <c-list>

c-expr:= column-name@table-name@schema-name | {<c-list>} |

<c-expr> OPR <c-expr>

Following is the list of derivation primitives for column integration, where every c_i stands for *column-name@table-name@schema-name* and c stands for *derived column@table-name*, union primitive is represented by "*{,}*", and extraction primitive is represented by "*OPR*":

1. *Column Rename*

   c = c_1

2. *Column Union*

   c = $\{c_1,..,c_n\}$

3. *Column Extraction*

   c = $c_1 OPR c_2 OPR,..c_m$ where *OPR* can be any type of arithmetic operation

   if c_i 's are of type numeric, and of string operation if c_i 's are of type string. The right and left hand side of the operation must be the same type.

As stated earlier, results of the schema integration process are stored in SDML. SDML uses the derivation constructs introduced above in order to specify and store the derivation history. An example for each type of derivation is provided below to show how these derivation constructs are used. Only the related part of the XML document is shown in the examples.

- *Table Rename Derivation* - renames a table and 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:type="TABLE"
             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. An example is given below.

```
<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>
```

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

```
<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 criteria (predicate). An example is given below.

```
<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 and 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. An example is given below.

```
<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:type="COLUMN"/>
    </graph:columnRenameDerivation>
</graph:snode>
```

- *Column Union Derivation* - is used to specify that a column in the integrated schema is the union of two 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:type="COLUMN"/>
        <graph:derivationNode graph:name="phoneno" graph:table="contact"
            graph:id="urn:sasmint:column:targetsc:contact:phoneno"
            graph:schema="targetsc" graph:type="COLUMN"/>
    </graph:columnRenameDerivation>
</graph:snode>
```

- *Column Extraction Derivation* – is used to specify that a column either in recipient or donor schema equals *n* columns of the other schema, which are combined using an arithmetic or string operation, such as concatenation. Currently, one type of *Column Extraction Derivation* is defined, called "*columnStringAdditionDerivation*", which is used to define that a column in one schema equals the concatenation of two or more columns in the other schema, as exemplified 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:type="COLUMN"/>
        <graph:derivationNode graph:name="intname" graph:table="contact"
            graph:id="urn:sasmint:column:targetsc:contact:intname"
            graph:schema="targetsc" graph:type="COLUMN"/>
        <graph:derivationType
            graph:refDerivationNode="urn:sasmint:column:targetsc:contact:intname">
            <graph:columnStringAdditionDerivation>
                <graph:derivationNode graph:name="lname" graph:table="contact"
                graph:id="urn:sasmint:column:targetsc:contact:lname"
                graph:schema="targetsc" graph:type="COLUMN"/>
                <graph:derivationNode graph:name="fname" graph:table="contact"
                graph:id="urn:sasmint:column:targetsc:contact:fname"
                graph:schema="targetsc" graph:type="COLUMN"/>
            </ graph:columnStringAdditionDerivation>
        </graph:derivationType>
    </graph:columnUnionDerivation>
</graph:snode>
```

4.2.5.2 Schema Integration Phase of SASMINT - *Rules*

Automatic integration of two relational schemas is challenging and not straightforward. When applying different cases resulted from the schema matching stage of the SASMINT, while for the majority of cases we have introduced automatic rules for their schema integration, there are still a few cases left for which the automation is not supported by our system at this stage, and therefore are left to the users to decide on how to perform their integration. These cases typically represent a large difference in the semantics applied by the designers of the donor

and recipient schemas, for which its complete automation would represent selecting only one integration option among many, which may not be the best solution.

Considering that the two schemas are labeled as the donor and the recipient, Table 4.7 represents the variety of cases of schema matching results that can be produced. For each case, a description is also provided, and then it is indicated whether the case is or is not supported by the introduced automatic schema integration approach of SASMINT. As listed in Table 4.7, the eleven match result cases given in 1, 2, 3, 5, 7, 9, 10, 11, 12, 13, and 14 are considered for automatic schema integration in SASMINT. The remaining match result cases (namely: 4, 6,

Table 4.7. Different possibilities after schema matching

Case	Match Result	Case description	Automated Integration
1	Column X (1 → 1) Column Y	Column X in the recipient schema matches Column Y in the donor schema	Applied
2	Column X (1 → n) Column	Column X in the recipient schema matches n columns of donor schema	Applied
3	Column X (1→ 1) Table A	Column X in the recipient schema matches Table A in the donor schema	Applied
4	Column X (1 → n) Table	Column X in the recipient schema matches n tables of donor schema	Not applied
5	Column (m → 1) Column Y	m columns of the recipient schema match Column Y in the donor schema	Applied
6	Column (m → n) Column	m columns of the recipient schema match n columns of the donor schema	Not applied
7	Column (m → 1) Table B	m columns of the recipient schema match Table B in the donor schema	Applied
8	Column (m → n) Table	m columns of the recipient schema match n tables of the donor schema	Not applied
9	Table A (1→ 1) Table B	Table A in the recipient schema matches Table B in the donor schema	Applied
10	Table A (1→ n) Table	Table A in the recipient schema matches n tables of the donor schema	Applied
11	Table A (1→ 1) Column Y	Table A in the recipient schema matches Column Y in the donor schema	Applied
12	Table A (1→ n) Column	Table A in the recipient schema matches n columns of the donor schema	Applied
13	Table (m → 1) Table B	m tables of the recipient schema match Table B in the donor schema	Applied
14	Table (m→ n) Table	m tables of the recipient schema match n tables of the donor schema	Applied
15	Table (m→ 1) Column Y	m tables of the recipient schema match Column Y in the donor schema	Not applied
16	Table (m→ n) Column	m tables of the recipient schema match n columns of the donor schema	Not applied

8, 15, and 16) correspond to certain highly complex integration cases, for which an automatic solution is not advisable by SASMINT approach and therefore it will not be applied for them.

In order to automatically generate integrated schema based on the results of schema matching, a number of heuristic rules are defined in SASMINT. These rules cover the cases marked as "Applied" in Table 4.7.

Integration process starts with table matches and continues with column matches. This means the match pairs, of which one side is a table, are processed first. All tables that do not appear in any match pair, as well as all their non-matching columns are directly added to the (recipient) integrated schema. The first five rules in table 4.7 are related to table names. Rules 6, 7, 8, and 9 are related either to the table-to-column or to column-to-table matches. After the first nine rules are applied, non-matching columns of all tables are checked one more time in order to see if they are all already covered in the integrated schema. This check is handled by Rule 10. All non-matching columns of tables that are not yet covered at this stage will then be directly added to the integrated schema. After this step, rules 11, 12, and 13 are applied only to the column-to-column match results. More details about the schema integration rules are provided below.

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} 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}

For each column of *m* tables, $T_{d1..dm}$, of the donor schema 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 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 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

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. Its algorithm is represented as follows:

Begin

Generate a new table node, T_{i1}, based on the table T_{d1} of the donor schema and add it to the integrated schema.

For each column of T_{d1} 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}

For each column of *m* tables, $T_{r1..rm}$, of the recipient schema 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 operator to specify that the newly generated table, T_{i1} is the union of T_{d1} and $T_{r1..rm}$. Include this derivation in the integration result.

Apply the Column Rename Derivation 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_{d1} and $T_{r1..rm}$.

End

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. 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} 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}

For each column of T_{d1} 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 operator to specify that the newly generated table, T_{i1} is the union of T_{r1} and T_{d1}. Include this derivation in the integration result.

Apply the Column Rename Derivation 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}.

End

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 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 m tables, $T_{r1..rm}$, of the recipient schema, 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}

For each column of *n* tables, $T_{d1..dn}$, of the donor schema, 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 operator to specify that the newly generated table, T_{i1} is the union of $T_{r1..rm}$ and $T_{d1..dn}$. Include this derivation in the integration result.

Apply the Column Rename Derivation 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 $T_{r1..rm}$ and $T_{d1..dn}$.

End

Rule 5: This rule applies to the tables that are not involved in any match pair and 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 $T_{d1..dn}$ in recipient and donor schemas respectively

For each $T_{r1..rm}$ and $T_{d1..dn}$

Generate a new table, T_{ix} *and add it to the integrated schema*

For each column of $T_{r1..rm}$ and $T_{d1..dn}$, 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 to specify that the newly generated tables are the renamed versions of $T_{r1..rm}$ and $T_{d1..dn}$.

Apply the Column Rename Derivation 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 $T_{d1..dn}$ of 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 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 to specify that the newly generated column of the integrated schema is the union of this column and m columns ($C_{d1..dm}$) *of the donor schema*

Apply Table Rename Derivation 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 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 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 to specify that the newly generated column of the integrated schema is the union of this column and column (C_{d1}) *of the donor schema*

Apply Table Rename Derivation to specify that the newly generated table is the renamed version of table T_{r1}.

End

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. Its algorithm is represented as follows:

Begin

Generate a new table node, T_{i1}, based on the table T_{d1} of the donor schema and add it to the integrated schema.

For each column of table T_{d1} of the donor 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_{d1} of the donor schema, which is added to the integrated schema in the previous step

Add Column Union Derivation to specify that the newly generated column of the integrated schema is the union of this column and m columns ($C_{r1..rm}$) *of the recipient schema*

Apply Table Rename Derivation to specify that the newly generated table is the renamed version of table T_{d1}.

End

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. Its algorithm is represented as follows:

Begin

Generate a new table node, T_{i1}, based on the table T_{d1} of the donor schema and add it to the integrated schema.

For each column of table T_{d1} of the donor 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_{d1} of the donor schema, which is added to the integrated schema in the previous step

Add Column Union Derivation to specify that the newly generated column of the integrated schema is the union of this column and (C_{r1}) *of the recipient schema*

Apply Table Rename Derivation to specify that the newly generated table is the renamed version of table T_{d1}.

End

Rule 10: This rule applies to the columns of tables that are not involved in any match pair and all such columns that do not match anything are directly added to the integrated schema. 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 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 to specify that the newly generated column C_{i1} is the union of C_{r1} and $C_{d1..dm}$

If Column String Addition Derivation is defined, apply this integration rule 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

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 represented as follows:

Begin

Identify the parent table T_{d1} of the column C_{d1} in the integrated schema. Search all table derivations to find out where this table T_{d1} exists and identify the related table T_{i1} in the integrated schema.

Generate a new column C_{i1} based on the C_{d1} and add it to the integrated schema as the column of T_{i1}.

Check whether C_{d1} 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 $C_{r1..rm}$ of the recipient schema.

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

If Column String Addition Derivation is defined, apply this integration rule 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_{d1} and C_{x1}

End

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 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}.

Apply Column Union Derivation to specify that the newly generated column C_{i1} is the union of C_{r1} and C_{d1}.

End

4.2.5.3 Schema Integration Phase of SASMINT - *Result Generation*

The result of schema integration phase of SASMINT is displayed both in graph format and in SDML format. The automatically generated integrated schema is shown as a DAG, while the SDML representation of the result formally shows which elements of the input schemas are represented using which element of the integrated schema and using what kind of a derivation this new element of the integrated schema is generated.

4.2.6 User Modification and Validation Phase – *P5*

Since schema integration is a challenging process, user modification and validation are also necessary after the automatic generation of the integrated schema, as represented in Figure 4.3. Therefore, users' validation and/or modification of both the integrated schema and the derivation results at this stage guarantee the success of the automated schema integration process. Similar to the User Modification and Validation Phase after schema matching, GUI resides at the heart of this phase, to better facilitate users' validations and modifications.

Schema integration in SASMINT enables iterative development of a global integrated schema for a network of databases within a collaborative network environment, through the integration of two schemas at a time. Namely, in any network, first, the schemas S_1 and S_2 of two nodes are selected by the user and identified as donor and recipient. After the completion of schema matching, S_1 and S_2 get integrated. This result is displayed in graph as well as in SDML format. Then the user will have the opportunity to check and validate these results, or apply any necessary modifications.

In principle, the user applies the required modifications on the SDML representation of the result. User modification is required in two cases:

1) When *automatic schema integration is not performed by SASMINT:* For example, for the match result of "Column X (1 → n) Table", no integration rule is defined in SASMINT, and thus this type of match is not processed by the automatic schema integration, which needs human interference.

2) When *the result generated by SASMINT is not valid:* Since SASMINT generates the integrated schema based on the schema matching results, if there are any mistakes in those

results that the user has missed to correct after schema matching phase, then the schema integration phase may produce invalid results. In this case, the user is advised to back track and make any necessary corrections on the schema matching results and repeat the schema integration phase, unless the user is certain about the needed correction and wishes to directly make the corrections on the SDML representation of the integrated schema.

After applying all desired modifications on the integrated schema generated by SASMINT, the user can save the result, which corresponds to $S_{\text{int}1}$ for S_1 and S_2. The $S_{\text{int}1}$ will then become the new recipient schema for any further integration. Namely, to continue with generation of a globally common schema for a collaborative environment, the user (schema integrator) may then select $S_{\text{int}1}$ and another donor schema S_3 (from another node) and repeat the process to integrate them into $S_{\text{int}2}$. This process continues until all schemas from the network nodes are integrated, resulting in a final global integrated schema S_{int}.

4.3 Conclusion

Providing infrastructures for supporting data sharing among heterogeneous, distributed, and mostly autonomous databases has been an important open research question in the database research area. We categorize the related studies under four groups: **1)** studies focusing on database integration and interoperability problems, **2)** studies focusing on schema matching, **3)** studies focusing on schema integration, and **4)** studies focusing on ontology matching and ontology merging.

Most current research addresses one problem area and suggests solution that typically requires large amount of manual work. In order to resolve heterogeneity and enable semi-automation of both matching and integration of schemas, and to support interoperability and data sharing within networks of databases, we propose the SASMINT approach. As for the target application problem space, SASMINT is applicable to different types of applications and purposes, as addressed in Section 4.2, including: 1) database federation with a common schema, 2) full database federation, and 3) incremental generation of an integrated global schema.

In the introduced SASMINT approach, covered in this chapter, the following main goals have been addressed and discussed:

- *Using a Combination of Applicable Match Algorithms*: A combination of linguistic and structure matching algorithms from the NLP and Graph Theory domains are addressed.
- *Enabling Semi-Automatic Schema Integration*: Using the results of schema matching for the purpose of automatically generating an integrated schema.
- *Providing a Graphical User Interface*: An intuitive GUI for assisting the users with the process of manual intervention in cases where automatic conflict resolution is not possible.

For the SASMINT approach to achieve these goals, it introduces five main phases: 1) Configuration Phase, 2) Automatic Schema Matching Phase, 3) User Modification/Validation (of match results) Phase, 4) Schema Integration Phase, and 5) User Modification/Validation (of integration results) Phase.

We showed in this chapter that the SASMINT approach is more comprehensive and can produce more accurate results, when compared to previous approaches for data sharing among heterogeneous databases. In addition to using a combination of widely accepted matching algorithms, SASMINT also introduces the use of schema matching results for schema integration purposes. Other key innovations incorporated in our approach involve: 1) Introduction of the SAMPLER component for semi-automatic identification of the appropriate weights of the algorithms used in linguistic matching, 2) proposing an XML-based derivation language called the SDML, for formalization of SASMINT and capturing the results of both schema matching and schema integration processes.