On semi-automated matching and integration of database schemas

Ünal Karakaş, Ö.

Publication date
2010
Document Version
Final published version

ISBN 978-90-5776-218-5

Özgül Ünal Karakaş

On semi-automated matching and integration of database schemas

ON SEMI-AUTOMATED MATCHING AND INTEGRATION OF DATABASE SCHEMAS

Özgül Ünal Karakaş

On Semi-automated Matching and Integration of Database Schemas

Özgül Ünal Karakaş

The cover was designed by the author.

ISBN: 978-90-5776-218-5

On semi-automated matching and integration of database schemas

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. D.C. van den Boom
ten overstaan van een door het college voor promoties
ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op woensdag 24 november 2010, te 12:00 uur

door

Özgül Ünal Karakaş

geboren te Anamur, Turkije

Promotiecommissie

Promotor:	Prof. dr. H. Afsarmanesh
Overige Leden:	Prof. dr. B. J. Wielinga
	Prof. dr. L. Hardman
	Prof. dr. M. T. Bubak
	Prof. dr. L. M. Camarinha-Matos
	Prof. dr.-ing. B. R. Katzy

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Contents

List of Abbreviations

AWT	Abstract Windowing Toolkit
CML	Conceptual Modelling Language
CN	Collaborative Network
CNO	Collaborative Networked Organization
COIN	COntext INterchange
COMA	COmbination of MAtching algorithms
DAG	Directed Acyclic Graph
DBMS	Database Management System
DDB	Distributed Database
DDBMS	Distributed Database Management System
DDL	Data Definition Language
DL	Description Logic
FOAM	Framework for Ontology Alignment and Mapping
GRQ	General Research Question
GUI	Graphical User Interface
ICT	Information and Communication Technology
IDE	Integrated Development Environment
JWNL	Java WordNet Library
MOMIS	Mediator EnvirOnment for Multiple Information Sources
MSL	Mediator Specification Language
MSNF	Mediated Schema Normal Form
NLP	Natural Language Processing

NOM	Naïve Ontology Mapping
ODL	Object Definition Language
ODMG	Object Data Management Group
OEM	Object Exchange Model
OKBC	Open Knowledge Base Connectivity
ONION	ONtology composition
OWL	Web Ontology Language
PORSCHE	Performance Oriented SCHEma mediation
PROTOPLASM	PROTOtype PLAtform for Schema Matching
QOM	Quick Ontology Mapping
RDFS	Resource Description Framework Schema
RQ	Research Question
SDM	Semantic Data Model
SDML	SASMINT Derivation Markup Language
SF	Similarity Flooding
SIMS	Services and Information Management for decision Systems
TSIMMIS	The Standford-IBM Manager of Multiple Information Sources
UML	Unified Modeling Language
VO	Virtual Organization
XDR	XML Data Reduced
XMI	XML Metadata Interchange
XSD	XML Schema Definition

Chapter 1

Introduction

To effectively use and benefit from the vast amount of information provided online by a large number of databases, this information needs to be interlinked and integrated. This requirement has become more evident with the increasing demand for remote collaboration among independent organizations and individuals. An important first step in this direction is to support the matching of independently developed meta-data in database schemas of different organizations and individuals. This needs to be done through resolving variety of their heterogeneities, and identifying correspondences among concepts defined in these database schemas. Furthermore, for proper interlinking of these databases, another necessary step is the integration of their database schemas. Resolving these complexities is challenging and quite inefficient to handle manually. The thesis proposes an automated but supervised approach, called SASMINT- Semi-Automatic Schema Matching and INTegration, which addresses and merges the problems of matching and integration of relational database schemas. This chapter provides some introductory information about the research work carried out towards provision of the proposed approach. Section 1.1 addresses the motivation for this research. Section 1.2 enumerates the main research questions, followed by the main objectives and contributions of the research addressed in Section 1.3. Section 1.4 specifies the scope of this research. Finally, Section 1.5 elucidates the applied research method, and Section 1.6 outlines the structure of the thesis.

1.1 Motivation and Requirements Analysis

Advances in information and communication technology (ICT) have created new opportunities for computing world-wide. High speed networks enable us to reach large quantities of information within fraction of seconds. However, these developments create many new challenges. One such example is how to link and share large amounts of similar or inter-related data provided by distributed, heterogeneous, and autonomous parties who wish to work with each other.

The importance of developing a support infrastructure for data sharing has been addressed and understood clearly during the last decade, with the increasing need for collaboration among organizations. The term collaboration among organizations is now used frequently, as defined in (Camarinha-Matos & Afsarmanesh, 2008b):

> **Collaboration** is a process in which entities share information, resources and responsibilities to jointly plan, implement, and evaluate a program of activities to achieve common goals.

For companies for instance, in order to remain competitive in a highly aggressive global market, they need to become more agile in coping with changes and achieve this goal in a better and faster manner. As a response to this challenge, Collaborative Networks have emerged.

> A **collaborative network** (CN) is an alliance constituted by a variety of entities (e.g. organizations and people) that are largely autonomous, geographically distributed, and heterogeneous in terms of their operating environment, culture, social capital and goals, but that collaborate to better achieve common or compatible goals, and whose interactions are supported by computer networks (Camarinha-Matos & Afsarmanesh, 2008a; Camarinha-Matos et al., 2005).

Several forms of collaborative networks are currently observed, as shown in Figure 1.1. The two top level collaboration forms in this classification are the ad-hoc collaboration and Collaborative Networked Organizations (CNOs). The ad-hoc collaboration represents formation of spontaneous collaborations, without any predefined goal, such as the instantaneous on the spot formation of a rescue collaboration team to assist with a disaster. On the other hand, CNOs are carefully established with participating organizations, having different roles in the network, towards achieving their common goals. There are also two

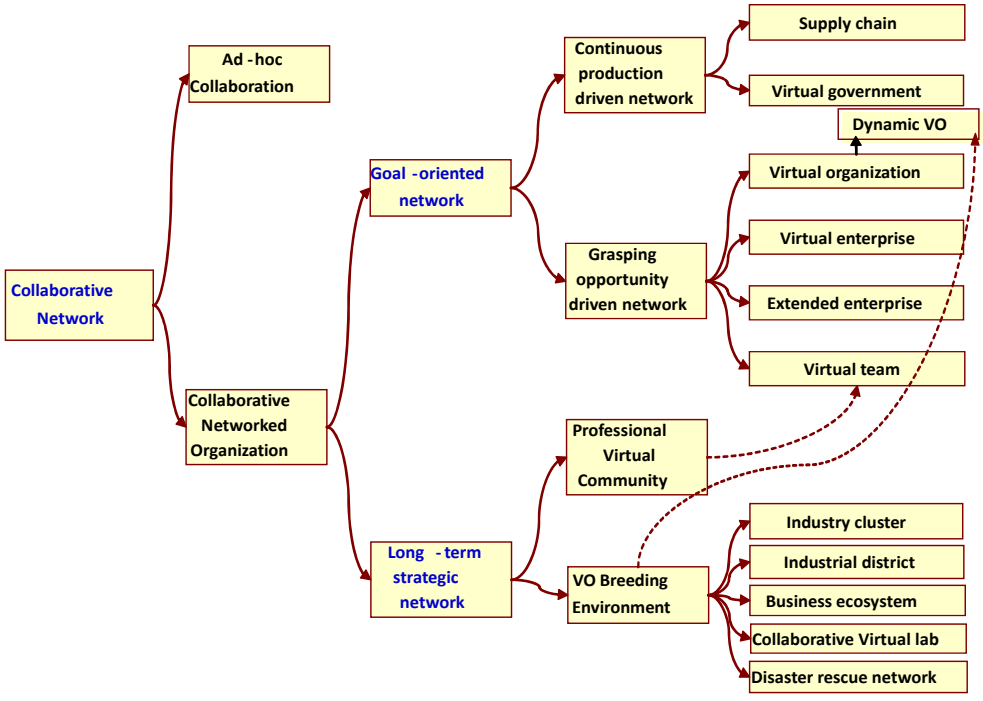


Fig. 1.1. Different forms of collaborative networks (Camarinha-Matos & Afsarmanesh, 2008a)

forms of CNOs called goal-oriented and long-term strategic networks. Members of a goal oriented network work together to achieve their common goals, whereas long-term strategic networks are strategic alliances aimed to prepare their member organizations towards dynamic establishing of focused collaborative networks at the emergence of new opportunities in the market/society. Goal-oriented networks can be either continuous production driven networks or grasping-opportunity driven networks, while long-term strategic networks can be either professional virtual community or virtual organization breeding environment (Afsarmanesh & Camarinha-Matos, 2005). At the lowest level of classification, supply chain, virtual government, virtual organization, virtual enterprise, extended enterprise, virtual team, industry cluster, industrial district, business ecosystem, collaborative virtual laboratory, and disaster rescue network represent the main variety of types of networks that are manifested today in parallel. Details about each of these types of collaborative networks are provided in (Camarinha-Matos & Afsarmanesh, 2008a).

We take Virtual Organizations as an example. Briefly, a *Virtual Organization* (VO) is a gathering of autonomous organizations through a network that pursue the accomplishment of a set of specific common goals (Camarinha-Matos et al., 2000). There are a number of benefits associated with VOs (Afsarmanesh & Camarinha-Matos, 1997), including the increased access to market/society opportunities, sharing risks, reducing costs, and achieving business/societal goals not achievable by a single organization and thus the motivation for involvement in VOs. A VO represents a complex and dynamic entity that undergoes a sequence of stages during its life cycle (Camarinha-Matos & Afsarmanesh, 1999a; Camarinha-Matos & Afsarmanesh, 1999b), as shown in Figure 1.2.

In all stages of the VO lifecycle, ICT support is needed. For example, in order to enable rapid formation of VOs, it is required to establish a common interoperable infrastructure (Afsarmanesh & Camarinha-Matos, 2005) that is typically achieved within the VO breeding environments (Afsarmanesh et al., 2008). Members of VOs need to strongly interact with each other to achieve the goals of the VOs and one form of this interaction is by means of data

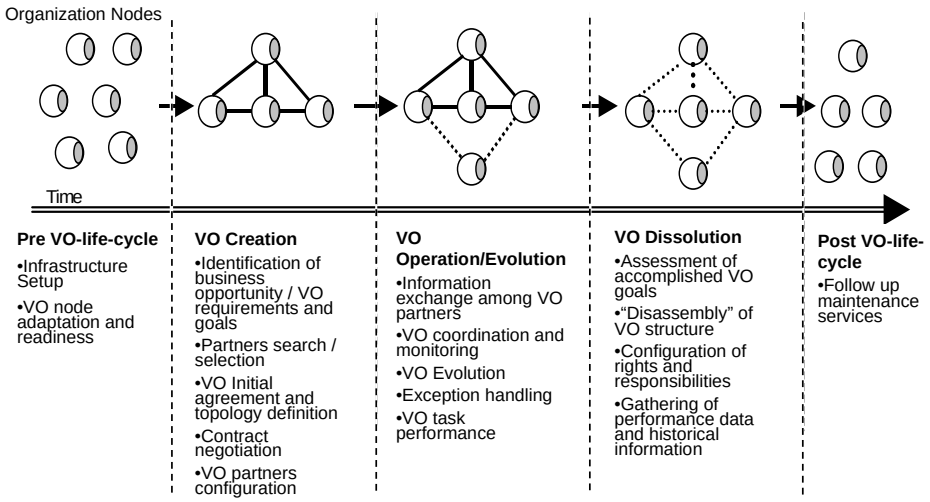


Fig. 1.2. VO Life Cycle

sharing, which requires information integration and interlinking.

Although in practice the first examples of collaborative networks come from the manufacturing domain, the need for collaboration has been well understood recently in different domains, such as engineering, economy, social sciences, etc. One such example domain is the biodiversity. Increasing number of biodiversity conservation activities entail producing more accurate results by comparing and/or merging different biodiversity data analysis results to make better predictions about the global status and distribution of species, as well as protection of those who are endangered.

This, in turn requires the collaboration and data resource sharing among many worldwide distributed biodiversity centers, organizations, and individual researchers (Unal & Afsarmanesh, 2006a; Unal & Afsarmanesh, 2006b; Unal & Afsarmanesh, 2006c; Unal & Afsarmanesh, 2009; Unal & Afsarmanesh, 2010). Although the importance of collaboration in biodiversity has become clear to most involved scientists, so far most biodiversity related organizations hesitate to actively cooperate. This is mostly due to the sensitivity of some specific data categories, such as those related to endangered species in biodiversity domain, where there is the danger of unintentionally creating new business opportunity for illegal poachers, though announcing the information about species in danger of extinction. Therefore, new mechanisms and infrastructures are needed, supporting information sharing among organizations, while taking the needed criteria into account. With the existence of such mechanisms, organizations can more easily decide to collaborate.

There are however some difficulties facing the infrastructures to support data sharing and exchange in the biodiversity domain as follow. Different organizations structure biodiversity data in different formats depending on their specific needs and preferences. The level of detail that they keep for their managed data also greatly differs. They typically do not use standard data models and different syntax is used. Likewise, since different centers use their own controlled terminology and vocabularies, the semantic definitions of their data and used concepts are heterogeneous. These matters have resulted in a large number of independent and heterogeneous databases, scattered all over the world. Because of the differences in managing their data, these databases are quite rarely integrated effectively. Furthermore, any effort spent on such integration is usually not at the global network level, rather bi-lateral focused on each pair of databases. Hence, demands for an effective uniform mechanism to integrate/interlink a number (possibly large number) of databases, to support homogeneous access to heterogeneous and distributed databases, thus providing a single and integrated interface for users in biodiversity networks are increasing.

As discussed above, in biodiversity domain, collaborating organizations need to share data with others and simultaneously access and manipulate data from others, and thus integration of data is required. Among other challenges, the heterogeneity, distribution, autonomy, continuous and rapid technologic evolution, and multi-disciplinarity of the area are the main obstacles faced to achieve the required integration in all levels of collaborative networks (Camarinha-Matos & Afsarmanesh, 2003), including those of the biodiversity. We can further elaborate on these common obstacles as follows: 1) *Heterogeneity*: It arises due to the lack of common standards, as each organization uses its own format and model for database schema definitions, which makes the interoperation among nodes much more difficult. Especially if the number of organizations in a CN is large, an important challenge is how to share and integrate data represented by heterogeneous database schemas. 2) *Distribution*: Organizations are logically and physically distributed. However, improvements in the field of high-speed networking help to decrease the impact of this obstacle and properly support remote access to distributed databases. 3) *Autonomy*: Organizations autonomously decide what to share and with which other organizations. Furthermore, each data owner in CNs is autonomously deciding on the representation and modeling of its data, which clearly and vastly increases the

heterogeneity. 4) *Continuous and Rapid Technologic Evolution*: Fast improvements in technologies lead to continuous changes in the format and amount of data to be exchanged. This evolution further increases the heterogeneity problems. 5) *Multi-disciplinarity*: Since each organization is from different types of areas and disciplines, integration and interlinking of information needs to handle wide variety of information types and their specificities. Design and implementation of an integration infrastructure for CNs, needs to take all these obstacles into account.

Heterogeneity is the most relevant obstacle among the others facing the data sharing for collaborations. The collaboration infrastructure has to consider such differences for providing effective mechanisms to integrate/ interlink and for homogeneous access to databases. Rather than accessing and manipulating single database systems in isolation for CNs, database research is needed to address simultaneous access and manipulation of different remote databases, as suggested in federated databases and multidatabase approaches (Hammer & Mcleod, 1979; Heimbigner & Mcleod, 1985; Sheth & Larson, 1990). However, automatic resolution of schema heterogeneity still remains as a major bottleneck for provision of integrated data access/sharing among autonomous, heterogeneous, and distributed databases.

Since each data owner in CNs decides autonomously on the representation and modeling of his data, data are typically and widely heterogeneous even if from the same domain. In order to provide transparent access to such remote data and enable the sharing of information among databases, their schema heterogeneity needs to be identified and resolved and then the correspondences among different organizations' schemas need to be identified. This process is called in research as schema matching. After schema matching process, to support collaboration, e.g. the possibility of processing federated queries within the network, schemas usually need to be integrated, to facilitate the needs of the CNs. It is clear that schema matching and integration constitute two key processes of the ICT infrastructure supporting the collaboration among organizations. Thus, tools that enable semi-automatic matching and integration are among the most important components of such infrastructures.

The most difficult part in resolution of heterogeneity of schemas during the schema matching process is the identification of the semantics introduced at each organization that are incorporated into their schema definitions. Data semantics are related to database designers' preference or interpretation of data, according to their understanding of the world within their organizations. Different interpretations cause different representations of data and thus different data models. In the process of comparing different database schemas for instance, semantic heterogeneity arises out of the ambiguity inherent in the separation between names (words) chosen in a data model and what they represent within the organization that originates them. Some semantics in general can be inferred from data, schema, and annotations if they exist. However, to put it simply, this information is most of the time incomplete and such inferences are not fully accurate, to decide whether an element *x* of a schema A matches an element *y* of another schema B and there is no other element *z* of this second schema B that matches *x* better than *y*. Therefore, a fully automatic solution for schema matching may not produce the best results.

Approaches so far proposed in database research for providing access to distributed, heterogeneous, and autonomous data sources have addressed some aspects of semi-automatic schema matching and/or schema integration in their approach. However, these approaches suffer from some or all of the following main limitations:

- No approach deals with both schema matching and schema integration together. Furthermore, it is generally not addressed how to formalize the result of schema matching and how to facilitate and support the needed semi-automatic schema integration.

- A fully automatic schema matching and integration is not realistic, considering that some types of semantic and structural conflicts are difficult to resolve automatically, as addressed later in Chapter 3. Therefore, a simple but effective user interface is needed to enable user interaction with the system for modification of the results generated automatically for both schema matching and schema integration. Nevertheless, in the proposed approaches so far, the provision of needed user-friendly interface as a part of the proposed architecture is typically skipped.
- As for schema matching, the suggested approaches typically represent at least one of the following drawbacks:
    - Although the aim is to automate or semi-automate the matching process, the currently proposed solutions generally require too much manual work.
    - A limited number of algorithms are so far implemented each focused on the automatic resolution of certain specific challenges related to either syntactic, semantic, or structural conflicts, and there are no comprehensive solutions suggested. As explained in Chapter 3 and addressed by most other work, syntactic, semantic, and structural conflicts constitute the main categories of heterogeneities that exist among database schemas. While observing both the nature of existing schemas, which generally consist of elements with different syntactic, semantic, or structural characteristics, and our test results presented later in Chapter 6, in each case, using a combination of some of these algorithms, which are suitable for different types of elements and domains, is necessary for achieving more accurate results.

There are a number of key requirements that an ICT infrastructure for CNs needs to address and support for enabling the data sharing and exchange among organizations. Namely, the base requirements listed below, must be met independent of any specific solution for data sharing (Guevara-Masis et al., 2004; Unal et al., 2005).

- Organizations should be able to preserve their autonomy when they join a collaborative network. They should be able to autonomously decide which part of their data and with which other nodes to share.
- New organizations should be able to join the networks easily, and dynamic evolution of the schemas, representing the shared data, should be supported.
- Database administrators should be supported with tools to semi-automatically generate mappings from each of the different schemas to the integrated schema.
- Organizations in CNs should easily collect data from others without needing to deal with the underlying heterogeneities of databases.

Two of the most important components of an ICT infrastructure, which meet the requirements above for CNs, are the processes and components for schema matching and schema integration. Namely, the local schemas of a number of organizations need to be semi-automatically matched and integrated to generate a global schema for the CN.

Schema matching and integration play important roles in providing data sharing among distributed, autonomous, and heterogeneous databases. Taking into account the limitations of existing approaches, a comprehensive solution for semi-automatic schema matching and integration needs to focus on a number of specific requirements, as addressed below:

- **Semantic information needs to be identified:** Inherent in the schemas are large amounts of semantic information. Identifying semantic relationships is harder than simple relationships, as there are more possibilities that need to be taken into account. While observing the explicit relationships among schema elements, identifying implicit relationships is a problem that makes the automatic detection of elements' correspondences difficult. Auxiliary resources, such as linguistic dictionaries consisting of some semantic relationships among concepts, need to be utilized to identify as much semantic information as possible.
- **Both simple and complex matches need to be considered:** Most matching approaches limit their search to only one-to-one (1-to-1) matches (e.g. "email" to "electronic_mail"), also called as simple matches. Complex matches (e.g. "address" to "street", "zipcode", and "city") are much more difficult to identify than 1-to-1 matches. Although it is not realistic to extract all variations of matches automatically, at least complex matches in form of 1-to-n and n-to-1 need to be also identified to the extent possible.
- **Combination of a number of matching algorithms needs to be considered:** Schemas in general consist of element names in different formats. Some similarity algorithms produce better results when applied to certain specific types of element names. Therefore, it is not effective to pre-select and use only one or a few comparison algorithms, which are each suitable for certain types of names, for all kinds of schemas.
- **A Supporting user friendly Graphical User Interface (GUI) needs to be provided:** Developing only algorithms for automatic schema matching alone is not sufficient. User interaction is an important part of the process to be considered when developing the schema matching and schema integration systems. Especially considering that it is not possible to identify all matches automatically, a user-friendly and effective user interface is required to enable users' modification of the matched results. Furthermore, the results of schema integration also need final users' validation.
- **Schema matching process needs to be combined with schema integration process:** Schema integration, a challenging process especially considering all the conflicts that need to be resolved before the integration starts, requires at its base the identification of the correspondences among the source and target schemas, resulted by the schema matching. Therefore, a schema integration approach should facilitate the schema matching process by formalizing its user validated results and applying these results to the schema integration process. Proposing such a semi-automated schema integration approach and implementing it as a system provides a significant contribution to the information sharing and integration within the CNs.

1.2 Addressed Research Questions

In Section 1.1.1, we classified the general data sharing requirements for CNs under the umbrella of schema matching/integration. Namely, we addressed the CN's related requirements to support data sharing among distributed, autonomous, and heterogeneous databases. Addressing this problem area, we aim at developing formally founded and empirically validated approach and mechanisms. As such the main General Research Question (GRQ) for this thesis constitutes:

> ***GRQ-*** *How can we effectively and semi-automatically achieve the schema matching and schema integration, to facilitate data sharing in Collaborative Networks?*

We further refine this general research question into four specific Research Questions (RQs), which are addressed by this thesis.

In the first question, we address the terminology used in database research related to approaches and architectures for data sharing among heterogeneous data sources. This leads to the required understanding of the domain of our research problem:

> ***RQ1-*** *Which effective approaches and architectures can enable data sharing through interlinking and/or integrating heterogeneous databases of distributed nodes?*

Heterogeneity is the main problem to be tackled when dealing with schema matching and integration. It is therefore, necessary to differentiate the potential types of heterogeneities in order to identify those on which we need to focus during the schema matching and integration processes. This leads to our second research question:

> ***RQ2-*** *What is a representative taxonomy for addressing database schema heterogeneities, and in turn applicable to formalization of schema matching and schema integration challenges?*

Based on the state of the art and currently open research issues, we need to propose an appropriate approach for enabling semi-automatic schema matching and integration. This approach therefore should semi-automatically resolve different kinds of schema conflicts, such as the syntactic, semantic, and structural conflicts, in order to identify the potential matches among schema elements. The approach should be verifiable, e.g. a proof of concept as a working prototype of the system needs to be developed. Another important point that needs to be supported is the design of proper 'User Interaction'. These points lead to our third research question:

> ***RQ3-*** *What are effective mechanisms for semi-automatic schema matching and schema integration, and how should the user be involved in the process?*

We think the validation is important and necessary, in order to indicate the 'accuracy' and effectiveness of the approach we propose in comparison to other work. So, the final research question is built around the challenge of validating the developed system. The answer to this question shall reveal the appropriate measures that can be used for evaluating the accuracy of schema matching and schema integration, and thus the fourth research question constitutes:

RQ4- *How can we assess and validate the effectiveness of the proposed semi-automatic approaches for schema matching and schema integration?*

1.3 Objectives and Contributions of the Thesis

Aiming to address the main general research question described in Section 1.2, the main objective of this thesis is to propose an approach for resolving syntactic, semantic, and structural conflicts for semi-automatic schema matching and integration, facilitating data sharing and exchange in CNs. The answers to four specific research questions form the objectives of this thesis. Namely, the first research question (RQ1) is addressed by analyzing the related architectures and terminology used in database research for data sharing among heterogeneous data sources, which is the main subject of Chapter 2. The second research question (RQ2) is addressed by analyzing different taxonomies of heterogeneities proposed in the literature and defining the taxonomy of heterogeneity related to the challenges for schema matching and integration. This is the subject of Chapter 3. Our approach for semi-automatic schema matching and integration and its implementation are described in Chapter 4 and Chapter 5 in order to meet the research question three (RQ3). Research question 4 (RQ4) is addressed by carrying out validation against other related research. Chapter 6 describes this validation work and its results.

To conceptually verify our approach, we design and implement the SASMINT system that forms the basis for an infrastructure enabling users to query heterogeneous and distributed databases transparently in a federated database environment. Based on the proposed approach and its implementation, the main contributions of this thesis can be listed as follows:

- ✓ **Supporting both simple and complex matches:** Unlike many other approaches that support only simple matches, SASMINT supports both simple and complex matches, as addressed before.
- ✓ **Elevating the accuracy of schema matching:** In the SASMINT approach and implementation, we utilize a weighted combination of several schema matching algorithms. Syntactic, semantic, and structural conflicts are resolved by applying different specific string and structural similarity algorithms rooted in Natural Language Processing (NLP) and the Graph Similarity domains. Each algorithm best suits a specific type of strings and graph structures, and thus compounding some of them in SASMINT gives rise to more accurate matching results than other proposed approaches.
- ✓ **Enabling semi-automatic schema integration:** SASMINT interrelates directly the schema matching results with the schema integration. Heuristic rules are defined that run on the results of the schema matching and generate derivation formalism for an integrated schema automatically. We assess this as a novel contribution providing a strong competitive edge for the research on the SASMINT system.
- ✓ **Definition and incorporation of an XML-based language (an XML Schema) for enabling unambiguous interpretation of schema match / integration results:** Within the SASMINT system, we have devised an XML-based derivation language, which we call the SASMINT Derivation Markup Language (SDML) (in the format of XML Schema), that captures and supports the creation of a persisting schema

match and schema integration results**.** The value proposition of this particular contribution is multi-faceted. First, the persisted schema match integration results enable the external systems/agents to unambiguously interpret/understand the match / integration results. These external systems/agents could consume this information for implementing federated query processing, etc. Second, this generic format is understandable by the match/integration related human agents in-the-loop. What this means is that the human agents can then easily modify these results. Finally, the structure of the derivation language is designed to keep the derivation history. Namely for every entity, its entire derivation tree is preserved. This feature in turn enables the incremental schema integration procedure.

- ✓ **Enabling semi-automatic identification of suited weights for the composed algorithms:** A number of algorithms are utilized in the composite approach of the SASMINT system that calculates their weighted sum. Therefore, it is important to assign an appropriate weight to them, bearing in mind the suitability of each algorithm for different types of inputs. SASMINT provides the SAMPLER technique to semi-automatically identify the appropriate weights for the algorithms used in the linguistic matching.
- ✓ **Enabling user-friendly interaction by means of a GUI editor:** It is not possible to automatically extract all types of semantic and resolve all kinds of structural conflicts. Therefore, a suitable user-friendly GUI editor is provided for SASMINT for supporting the visual modification of the results of both schema matching and schema integration processes as well as their storage for further use.

1.4 Scope of the Research

There are different alternatives for representing database schemas. Besides using Data Definition Language (DDL) for relational database schemas, the XML Schema and the Web Ontology Language (OWL) are among the most popular representation mechanisms, and especially related to the increasing interest in the Semantic Web technologies. Since we focus on relational schemas in this thesis, we use the relational DDL for representing database schemas. Furthermore, for representing the integrated schemas, we use SDML, based on XML.

Since our aim is to semi-automatically match and integrate organizations' database schemas, and the relational database schemas are frequently used, the *relational schemas* constitute the main focus of our research explained as in this thesis. On the other hand, the proposed approach and the implemented SASMINT system is generalized and can be in principle extended to support other types of schemas, e.g. object-oriented as well. In the SASMINT system, relational schemas can be automatically loaded either from a relational database or from a previously saved XML file. When loading the schemas from a relational database, related metadata information, such as table and column names, is obtained from the database.

In general, the schema matching can consider different types of information as the base input, as explained in detail in Section 2.3.2. Our proposed solution however utilizes only the database schema related information (i.e. the metadata), and not the instance data. Instance data may not in general be available all the time, and using it might produce misleading and/or wrong results, if it is used alone, and without schema specification.

Different types of schema matches are addressed in Section 2.3.2. Our focus is on both simple matches (1-to-1 matches) and complex matches of type 1-to-n, n-to-1, and m-to-n.

1.5 Research Method

In the research for this thesis, we followed a method, composed of both theoretical and empirical work, as categorized in (Sørensen, 2005), which is in line with the standard scientific method.

An overview of the research phases is shown in Figure 1.3. A description of the steps in this approach is summarized below:

1. *Concept exploration and requirements analysis:* This phase constitutes the very first step of our followed method. Comprised within this step is an initial phase where an awareness of the concept of Collaborative Networks is achieved, together with an exploration of collaborative networks' data and information sharing related problems. What is further performed is an analysis of the ICT related information sharing requirements as well as the required supporting tools that would enable seamless data sharing within collaborative networks. The output of this phase is a basis, encompassing an awareness of: the collaborative networks, the data sharing requirements contained therein, the analysis of those requirements, and the resulting gap analysis, that are used for formulating the specific Research Questions that form the skeleton of our thesis work. In this step we come up with the finding that semi-automatic matching and integration of the database schemas used by the collaborating organizations is a very crucial step in solving their data interoperability/sharing related problems. We assess in this step that the resolution of this problem, i.e. the semi-automatic schema matching and integration, is definitely one main precondition to enable users with performing federated query processing transparently of its source databases over a network of collaborating entities.

2. *Identification and formulation of the Research Questions:* Based on the results of the previous step of the method, this step is where the main focus area of the thesis is devised and a context for the main research problem is established, around which the entirety of this thesis evolves. Upon an analysis of the requirements, the conceptual target area of 'schema matching and integration' is focused, and a list of research questions is built. Within the scope of the carried out research, we try to produce answers in this research for each research question listed under Section 1.2.

3. *Literature Survey and Review:* A thorough analysis of the existing theoretical and practical approaches that contribute to the resolution of the problems put forward within the context of the 'Research Questions' is performed at this step.

4. *Elaborating the Proposed Approach:* This step encompasses activities where we design a solution approach and a prototype that can be used as a supporting tool for matching and integration of heterogeneous schemas. Both the proposed approach and the prototype capitalize on the elicited data sharing requirements of collaborative networks. The scope and the objectives of our approach are refined in this step.

5. *Evaluation and also further validation of the proposed approach:* This step includes realization of the designed prototype in previous step that is used to evaluate and validate the proposed solution approach. The prototype is used to validate the adequacy of the research conducted for enabling the semi-automatic matching and integrating of database schemas from collaborative networked organizations.

Experimental evaluation of the accuracy of the proposed approach is also carried out in this step. Experiments are done using the prototype.

6. *Assessment of the results:* This phase unveils what sort of answers we have been able to produce for the research questions, and it subsumes an analysis of the answers produced. Also contained is an assessment of the value and contribution of the overall research work presented in this thesis in comparison to other related research in the area.

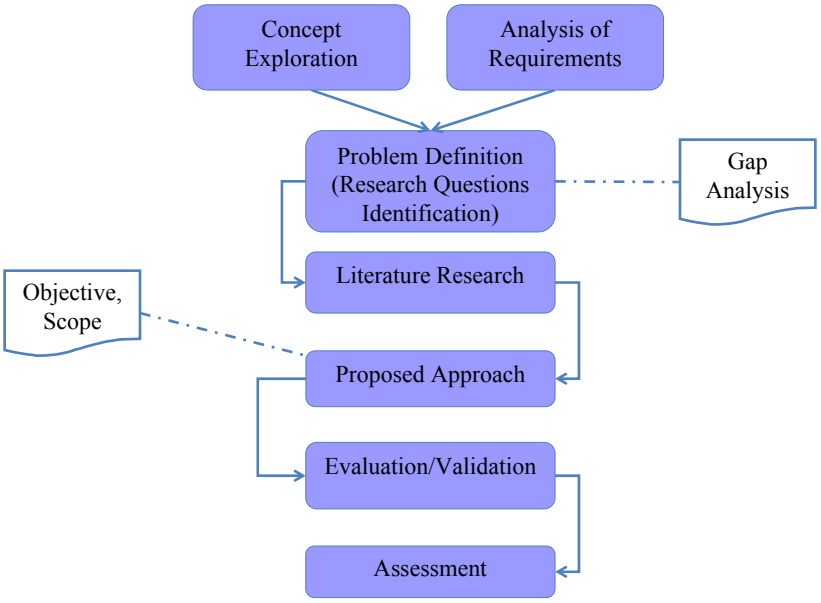


Fig. 1.3. Research method

1.6 Outline of the Dissertation

The rest of this thesis is organized as follows:

Chapter 2 provides different definitions presented in state of the art literature to refer to approaches, architectures, and systems for interlinking and/or integrating heterogeneous data provided by distributed databases in networks. Taxonomy of the terms related to an integrated information management system is provided in this chapter. Furthermore, the main features of schema matching and schema integration are addressed.

Chapter 3 aims at providing information about the heterogeneity as the most important problem to be tackled in infrastructures that enables data sharing. It addresses a number of heterogeneity (also called conflict) classifications, proposed in the literature. Furthermore, the

heterogeneity related challenges faced by the schema matching process are discussed by means of some examples.

Chapter 4 is dedicated to the SASMINT approach, proposed in the research work of this thesis. This chapter first starts with the related research, reviewing approaches focused on general database integration and interoperability, schema matching, schema integration, and ontology matching and merging. A number of open issues are addressed then to give a motivation for the proposed SASMINT approach. The rest of the chapter presents details about the phases of the SASMINT approach and how it achieves its goals.

Chapter 5 introduces the SASMINT system that is implemented to verify the approach proposed in this thesis. Details about the main components of the system are provided.

Chapter 6 provides information about the results of experimental assessment of the SASMINT system. Evaluation work covers schema matching, schema integration, as well as the Sampler components of SASMINT. Results of experiments comparing the schema matching approach of SASMINT and that of its closest competitor COMA++ are presented in this chapter.

Chapter 7 concludes the thesis with a summary of its contributions. It also presents the possible future improvements and next steps of this research.

The scientific publications related to the dissertation are listed in Appendix A.

Chapter 2

Interlinking and integrating schemas - background

Focusing on interlinking and/or integrating heterogeneous data from distributed nodes, database management research has introduced a number of approaches, architectures, and systems to enable their data sharing and data exchange, and in this process, it has also introduced a large variety of terms and concepts. This chapter addresses these approaches and definitions of these introduced terms and concepts that are closely related to schema matching and schema integration. Section 2.1 addresses these variety and it specifically represents our classification of the main concepts related to distributed information management, which are introduced in previous research. Section 2.2 depicts the main related categories of approaches from multidatabases research, based on schema coupling. Section 2.3 addresses the notions of schema integration and schema matching. Finally, Section 2.4 summarizes this chapter and emphasizes the importance of the automation of schema matching and schema integration processes.

The research results presented in this chapter were partially published in the Journal of Software (Unal & Afsarmanesh, 2009).

2.1 Related Concepts

High-speed networks have made it possible for the distributed information to be made available to everybody connected to the Internet. This has facilitated distributed information management systems, enabling access by authorized users to distributed data. The main requirements that need to be met with such systems have been summarized by (Kamel & Kamel, 1992) as follows: authorized users must be able to transparently access distributed and heterogeneous databases, there must be no changes needed in existing databases and applications, new databases should be easily added to the system, databases should be accessible for retrievals and updates, and finally performance of the system should be comparable to homogeneous systems.

Extensive research to enable data sharing in a distributed environment has given rise to variety of terms referring to different types of distributed information management systems. For instance, *distributed databases*, *multidatabases*, and *federated databases* are the most frequently used terms and concepts in the database research for several decades. However,

there are not yet commonly agreed definitions for these terms and concepts and quite often different researchers use the same term with different meanings. Therefore, the aim of this section is to provide enough background for this research, in order to differentiate among various definitions.

Distributed database (DDB) and distributed database management system (DDBMS) correspond to two base terms in distributed information management research. DDB and DDBMS are defined by (Ozsu & Valduriez, 1999) as follows:

> "A DDB is a collection of multiple, logically interrelated databases distributed over a computer network. A DDBMS is the software system that permits the management of the distributed database and makes the distribution transparent to the users."

In a typical distributed database management system, several databases over a network are managed by one management system. In other words, only one implementation of the database software is used in each network node.

According to (Ozsu & Valduriez, 1999), if the distributed database systems at various sites are also autonomous and possibly heterogeneous, they are referred to as *multidatabase* systems. Multidatabase systems allow integrated access to distributed, autonomous, and heterogeneous databases as (Bukhres & Elmagarmid, 1996) defines.

Multidatabase systems can be *homogeneous* or *heterogeneous*. Homogeneous database systems have the same database management systems and use the same data model and database manipulation language (Heimbigner & Mcleod, 1985) (Ozsu & Valduriez, 1999). Heterogeneous multidatabase systems on the other hand have different database management systems and use different data model and database manipulation language.

Another classification in *multidatabase* systems used by (Sheth & Larson, 1990) divides them into two types based on the autonomy of the participating database systems (components): *non-federated* and *federated*. Components in a *non-federated database system* are not autonomous. On the other hand, the components in a *federated database system* preserve their autonomy while also sharing their data in a partial and controlled manner. They share a part of their data by defining export schemas and making them available only to specific components. Every component is able to import schemas from other components according to the defined access permissions. As a consequence of this general interaction, this approach allows the cooperation between the nodes in the federation to accomplish a common or global task (Afsarmanesh et al., 2004).

(Sheth & Larson, 1990) categorizes federated databases further as *loosely coupled* and *tightly coupled*. A federated database system is loosely coupled if there is not a single authority to create and maintain the system; but this is the responsibility of users from each component system. There is no single global schema in loosely coupled systems. On the other hand, in tightly coupled systems, there is a central authority to administer the federation. If a tightly coupled federated database system only supports a single federated schema it is said to have a single federation, but if it supports multiple federated schemas it is said to have multiple federations. Users can submit queries applied to the federated schema, and the central authority is in charge of the distribution of sub-queries between the component databases and the processing of the individual results to satisfy the global request.

Besides terms referring to different types of distributed information management systems, another widely used term in database research domain is the *data integration.* Information systems mentioned above apply data integration techniques. *Data integration* aims at combining data residing at different sources and providing the user with a unified view of these data (Lenzerini, 2002). Data integration systems can be defined as a triple $\langle G, S, M \rangle$,

where G is the global schema, S is the heterogeneous set of source schemas, and M is the mapping between G and S. Two approaches are mentioned in the literature for defining M: *Global as View (GAV) (Chawathe et al., 1994)* and *Local as View (LAV) (Levy et al., 1996)*. In the GAV approach, there is a global schema expressed in terms of source schemas (S). Mappings M between the global schema G and source schemas S are well defined. However, when there is a new component database entering the system, a large amount of effort is required to update G. On the other hand, in the LAV approach, global schema is defined independently from source schemas and the relationships between the global schema and the sources are established by defining every source as a view over the global schema. Relationships between source schemas and the global schema may not be well defined here, which requires more complex query re-writing and thus puts more burdens on the query processor. Nevertheless, unlike GAV, addition of a new component database to the system does not require much effort.

Similar to variety of definitions related to distributed information management, there exist many definitions for database *interoperability*. For example, Brodie and Ceri (Brodie & Ceri, 1992) referred to interoperability as the ability of different systems to operate with each other. On the other hand, Silberschatz et al. (Silberschatz et al., 1990) defined interoperability as the problem of making heterogeneous and distributed databases behave as if they form part of a single database. Litwin and Abdellatif (Litwin & Abdellatif, 1986) and Zisman (Zisman, 1995) used the term interoperability to refer to the management and co-operation of multidatabase systems without using a global schema. Although there is no consensus on these definitions, database interoperability is a broader term than the terms related to distributed information management.

As it is clear from the definitions given above, there are many related terms concerning management of data provided by distributed and possibly heterogeneous and autonomous databases, whereas there is no consensus of terminology in the database community. In order to provide our understanding of the terms related to an integrated information management system, we have organized these definitions as shown in Figure 2.1.

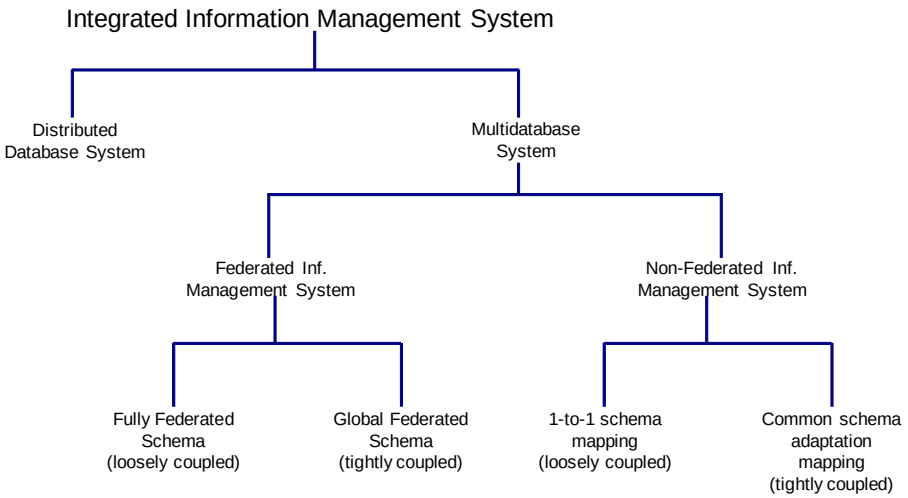


Fig. 2.1. Integrated Information Management System

Following the definition of (Ozsu & Valduriez, 1999), we mention two types of integrated information management systems: distributed database systems and multidatabase systems. Based on the classification of (Sheth & Larson, 1990), we divide the multidatabase systems as federated information management systems and non-federated information management systems.

Federated information management systems consist of autonomous nodes that can follow a fully federated schema or a global federated schema approach. As illustrated in Figure 2.2-a, a fully federated schema approach (Afsarmanesh et al., 1998) constructs an integrated schema at each node by merging the local schema of that node with the schemas imported from other nodes. Import schemas represent the information that other nodes make available to this node. A global federated schema approach on the other hand, generates a global schema by integrating the export schemas (representing the shared part of the information) from different nodes into a single schema, as shown in Figure 2.2-b.

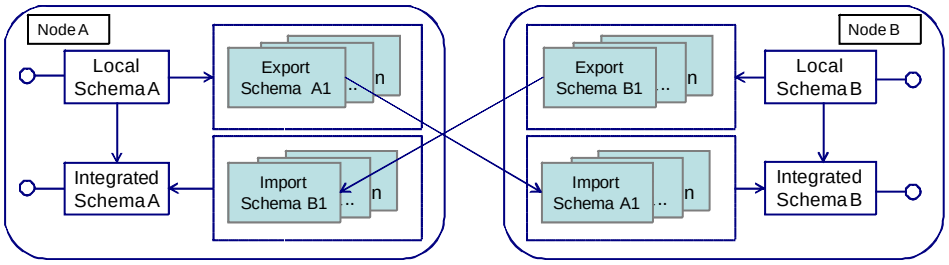


Fig. 2.2-a. Fully Federated Schema

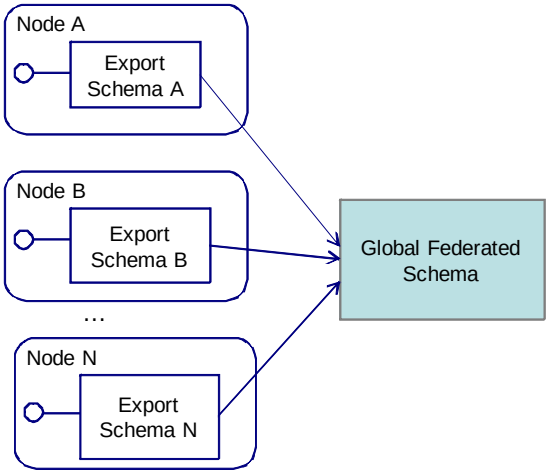


Fig. 2.2-b. Global Federated Schema

Nodes of non-federated information management systems are not autonomous. Two approaches can be mentioned here: 1-to-1 schema mapping and common schema adaptation mapping. In 1-to-1 schema mapping approach, mappings between the schemas of nodes are identified in a pair-wise manner. For instance, as represented in Figure 2.3-a, mappings between the schema of Node A and schemas of each other nodes are independently defined. Whereas in common schema adaptation mapping approach, mappings are specified between the common schema and the local schema of each node, as depicted in Figure 2.3-b.

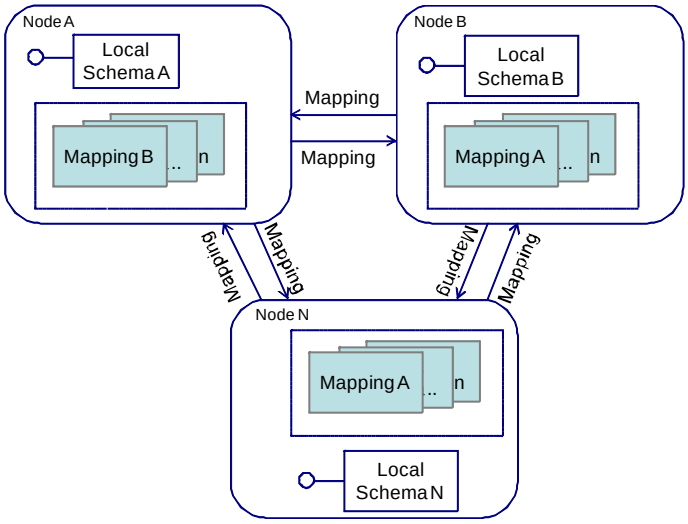


Fig. 2.3-a. 1-to-1 Schema Mapping

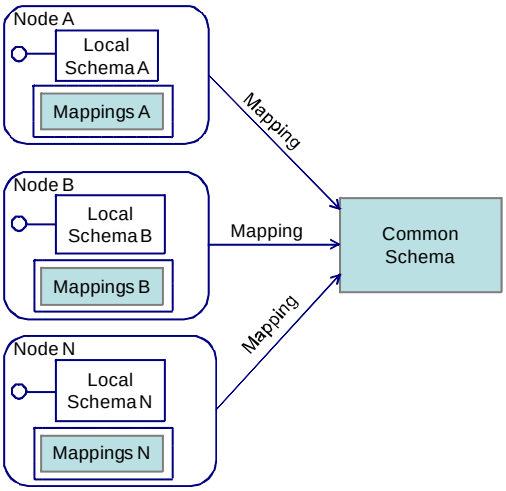


Fig. 2.3-b. Common Schema Adaptation Mapping

2.2 Multidatabase Classification Based on Schema Coupling

In this section, we focus on different types of multidatabase architectures based on schema coupling. We refer to multidatabase system as the one consisting of distributed and heterogeneous databases. By following definitions of (Zisman, 1995), we present a general overview of multidatabase architectures based on schema coupling in Figure 2.4.

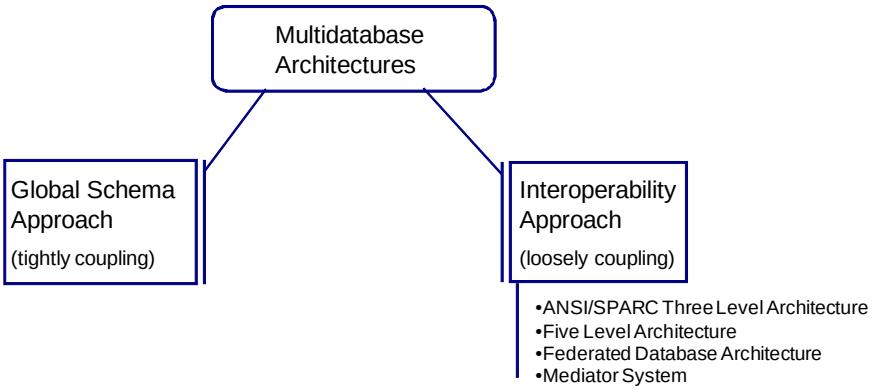


Fig. 2.4. A general overview of Schema Matching Approaches Based on Schema Coupling

In one of the multidatabase architectures, Global Schema Approach (also called as tightly coupling), there exists a single global schema representing all information across the databases. Global schema is generated by resolving the conflicts among local schemas and then integrating them into a single schema. A global schema is usually difficult to create as in order to create it one needs to fully understand the local database structures of participants. Generation of a global schema is achieved in several steps (Zisman, 1995). First, schemas are represented in a canonical data model, in case they are defined using different data models. Secondly, conflicts are resolved and the integrated schema is generated. In this approach, queries are created in terms of global schema and when such query arrives, it is decomposed into sub-queries to be sent to local databases. After this step, sub-queries are translated into the data language of the local database. When results of each query are received, they are merged for the final result to be sent to users. Global schema approach is suitable whenever the schemas are not subject to frequent changes. Advantages of this approach are; it is easy for querying and information loss is reduced. However, when the number of local schemas to be integrated is large or the environment is dynamic, this approach becomes complex.

In another multidatabase architecture, instead of creating a global schema, the aim is to make heterogeneous databases interoperable. In this architecture, either partial or no integration is required. Two types of interoperability approaches can be mentioned (Zisman, 1995):

1) *Direct Interoperability*, which consists of direct mappings (translations) among the components. Direct mapping is difficult when one schema is semantically more expressive than the other.

2) *Indirect Interoperability*, where an intermediate (canonical) data model and data manipulation language is used to manipulate heterogeneous databases. The canonical

data model is used to represent other models, bridge the gap between local models, detect inter-database semantic relationships, and achieve interoperability.

During the 80s, a variety of interoperability architectures were proposed in the literature, including ANSI/SPARC Three Level Architecture (Tsichritzis, 1981), Five Level Architecture (Sheth & Larson, 1990), federated database architecture (Hammer & Mcleod, 1979; Heimbigner & Mcleod, 1985), and mediator systems (Wiederhold, 1992).

Interoperability architecture overcomes the drawbacks of global integrated schema approach. This approach is appropriate when there are a large number of information sources and/or the environment is dynamic. However, the query processing costs are high in interoperability architectures.

An Example of Federated Database Architecture: PEER Federated Database System

The PEER system (Afsarmanesh et al., 1996; Tuijnman & Afsarmanesh, 1993) is a fully federated system designed and implemented at the University of Amsterdam. PEER is a generic object-oriented federated information management system enabling information sharing among autonomous and heterogeneous nodes. In the PEER architecture (see Figure 2.2-a), there is no need to create a global schema, as information stored in different nodes are interlinked through federated schemas. There are four types of schemas at each node: a local schema, a number of export schemas, a number of import schemas, and an integrated schema. The local schema models the local data at the node. Export schemas model the information that this node wants to share with other nodes of the network. Import schemas model the information that this node can access from other nodes. In other words, an import schema at each node is the export schema of another node that shares its data through this export schema. The integrated schema models the information that the node can access.

2.3 Schema Matching and Schema Integration

Organizations model their data using a variety of schema constructs. However, there is no single way to represent the same or similar data, which results in diversities in schema definitions even in the same organization. Distributed information management systems, introduced in the previous sections, need to tackle conflicts or heterogeneities and identify correspondences among schemas. As a result, schema matching and schema integration have become two main facilitating processes of distributed information management systems, which are mainly performed manually at present.

Schema specification is the main element of the schema matching and schema integration processes. A schema specifies how data is stored, accessed, and managed in the database management system (DBMS) and is described in a formal language supported by the DBMS. Examples of schema related languages include the SQL's DDL from relational data modeling domain, the Object Definition Language (ODL) from the object-oriented data modeling domain, the XML Metadata Interchange (XMI) of the Unified Modeling Language (UML), the XML Schema Definition (XSD) for XML documents, and the Resource Description Framework Schema (RDFS) as well as the OWL for ontologies.

The main inputs for the schema matching and schema integration processes are therefore the schemas. Following sub-sections make the role of schemas in these two processes clear. Also, more detailed information about schema matching and schema integration is given in these sub-sections.

2.3.1 Schema Integration

The problem of schema integration in the context of distributed information management systems is a relatively old challenge. In different approaches to enabling access to distributed and heterogeneous data, a different level of integration is achieved. Considering the classification of integrated information management systems, shown in Figure 2.1, schema integration is necessary in both fully-federated schema and global federated schema approaches. However, in the case of fully-federated schema approach, each node needs to integrate its local schema with the import schemas of other nodes to generate a representation of information that this node can access. On the other hand, in the case of global federated schema approach, schemas of all nodes, representing the information that these nodes make available to the network, need to be integrated to generate a single common schema that defines the information available at the network of nodes.

In database research, schema integration is typically used to refer to both the view integration and database integration (Batini et al., 1986). View integration aims at producing an integrated schema of users' views and is performed during the database design process, whereas database integration derives a new schema from existing specification. As identified in (Spaccapietra et al., 1992), view integration methodologies work with views based on the same data model, but database integration technologies work with schemas that are usually defined using heterogeneous data models. Considering the goals of the research work explained in this thesis, the focus is on the database integration. Therefore, when we use the phrase 'schema integration', we actually refer to integration of 'databases'. Furthermore, while we devise ways of semi-automatically integrating schemas, we target schemas which are based on the relational data model; i.e. both source and target schemas that we try to match and integrate are relational schemas.

There has been an extensive research work on the schema integration subject. A comprehensive survey of schema integration methodologies were done by (Batini et al., 1986). In (Batini et al., 1986), an analysis of twelve related methodologies were carried out, and they were compared based on different criteria, including the used data model, inputs, outputs, and strategies followed.

Considering all methodologies and approaches for schema integration and adding our own approach to it, three main integration steps can be identified, namely: 1) the Pre-integration step, 2) the Matching step, and 3) the Integration step.

1. The *Pre-integration* step consists of a number of preparation steps before the integration, such as identifying schemas to be integrated, preferences to be considered in the integration process, and amount of user input, as (Batini et al., 1986) mentioned. The type of the integration strategy followed affects the identification of schemas to be integrated. Two types of strategies are mentioned in (Batini et al., 1986) for schema integration: binary and n-ary strategies. Binary strategies allow the integration of two schemas at a time, while n-ary strategies can integrate *n* schemas at a time. Because of the complexities of integrating *n* schemas at a time, most approaches in the literature prefer a binary strategy.

2. The *Matching* step, also called the Investigation step by (Spaccapietra et al., 1992), identifies correspondences among different schemas by resolving their conflicts. Instead of the Matching step, the (Batini et al., 1986) categorizes two other steps called: comparison of the schemas and conforming the schemas, which together constitute the Matching step.

3. The *Integration* step is responsible for integrating the schemas, based on the correspondences identified in the matching step.

In this direction, the previously suggested schema integration approaches and methodologies are either fully manual or with some limited degree of automation focused only on the third step, and not including the matching step. Furthermore, for any automation on the integration step, it is typically assumed that the full semantics and structural knowledge of the two schemas are available.

Schema integration is a challenging and complex task: The integration step cannot be fully automated, since automatic resolution of some types of conflicts is not possible and user input is required to determine the appropriate meanings and decide on mappings for the integrated schema. Nevertheless, carrying out this process as automatically as possible and helping the users with this complicated task are needed in order to cope with the increasing demand for integrated information management systems.

The research work explained in this thesis addresses the full cycle of semi-automatic schema integration in three main steps, including: Configuration, Schema Matching, and Schema Integration, as shown in Figure 2.5. Some limited user input is required at these steps, as addressed below. The configuration step is responsible for assigning desired weights to the algorithms used in the linguistic and structure matching components, as well as for identifying the desired selection strategy for the results of schema matching. The schema matching step starts with a preparation activity that automatically turns the two source schemas (donor and recipient schemas) into a common format. Then, this process takes the schemas represented in the common format, as well as some other required inputs, as described in Section 2.3.2, and identifies all possible matches between the two schemas. After receiving the user input on the match results, at the third step, the schema integration takes as input the accepted correspondences between the two schemas and using a number of predefined integration rules, it automatically generates both an integrated schema as well as the needed mappings between the integrated schema and the two source schemas being integrated. The mappings are expressed in terms of a derivation language introduced in Chapter 4. Finally, the user input is required for the final validation of the integration results.

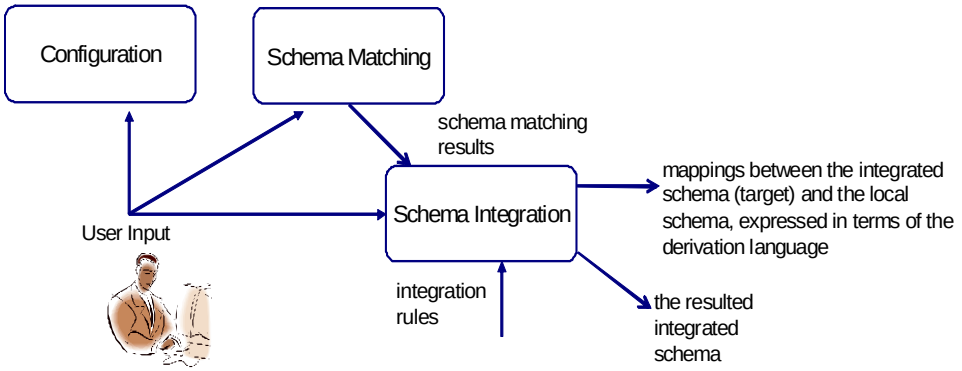


Fig. 2.5. Main Steps of the Schema Integration Process

2.3.2 Schema Matching

To achieve any of the integrated information management systems introduced in Section 2.1, there is a need to compare their schemas (e.g. two schemas at a time) and identify correspondences between them. As addressed in Section 2.3.1 on schema integration, federated information management approaches generate integrated schemas, where schema matching is one main step for the schema integration. In non-federated information systems however, the aim is to generate mappings either between the global schema for the network of databases and each of the local schemas - thus resulting a set of schema adaptation mappings, or between each pair of the local schemas - thus resulting a set of 1-to-1 schema mappings. Identifying the correspondences and generating the needed mappings also require support from the schema matching process.

Schema matching can be defined as the process for finding the correspondences between different elements of two schemas. The simplest type of matching is the 1-to-1 matching. For two schemas, e.g. A and B, schema matching process can identify for each element of schema A, the most similar element of schema B. In addition to 1-to-1 matches, some complex matches also frequently occur among schema elements. Complex matching identifies correspondences between each element or a group of elements of the schema A and a group of elements of schema B. Groups of elements are combined and inter-related with a formula. For example, suppose that there is a match identified between the 'name' element of Schema A and the 'fname' and 'lname' elements of Schema B. In this case, 'fname' and 'lname' can be combined through concatenation and a mapping can be defined between 'name' and this combination of 'fname' and 'lname'. Most schema matching approaches focus only on the 1-to-1 matches, considering that it is much easier to identify 1-to-1 matches than complex ones.

As shown in Figure 2.6, which represents a simplified version of the classification provided in (Rahm & Bernstein, 2001), individual and combined matchers are two top-level classes of matchers in this classification of schema matching approaches. Combined matchers represent a set of individual matchers. Individual matchers are further divided into two: instance-based and schema-based matchers. Instance-based matchers exploit the instance information; schema-based matchers on the other hand consider the definition of schema itself. Furthermore, schema-based matchers can be applied to individual schema elements (at element level) or for combination of elements (at structure level). Element level matchers use the linguistic characteristics of the element names. They apply techniques such as tokenization and word separation, removal of stops words and hyphens, expansion of abbreviations, and lemmatization, the details of which are all given in Chapter 4. Element level matchers consider both the syntactic as well as the semantic features of names in the schema. Furthermore, these types of matchers benefit also from the constraint-based techniques**,** which deal with the internal constraints being applied to the definitions of entities, such as types, cardinality of attributes, and keys.

On the other hand, the structure level matchers exploit the graph-based techniques. Graph matching techniques and the relationships among the graph elements together form the base of structure level matchers.

In (Shvaiko & Euzenat, 2005), a different classification is introduced, where three main dimensions are mentioned for the classification of schema matching algorithms, including:

1. *Input dimension*: This dimension is related to both the kinds of data or conceptual model to express schemas that the matching algorithms shall use, such as the relational or object-oriented, as well as the kinds of elements that algorithms shall exploit, such as the schema level and/or instance level data.

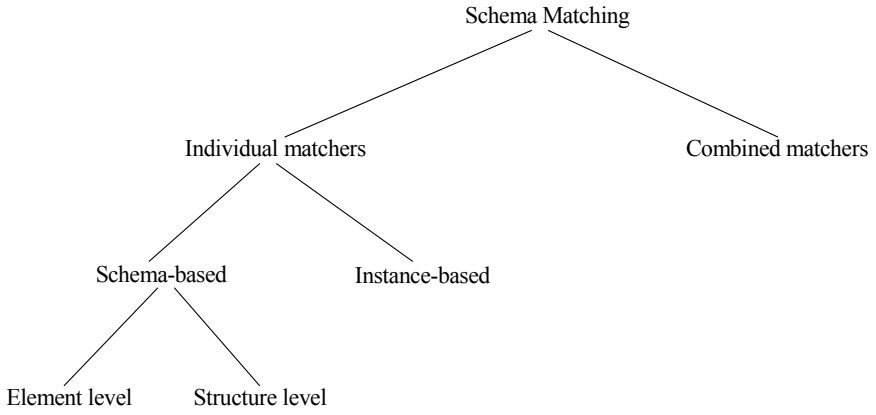


Fig. 2.6. A general overview of Schema Matching Approaches (simplified version of (Rahm & Bernstein, 2001))

2. *Process dimension*: This dimension considers the nature of the computation in the matching algorithms, which can be exact or approximate. For the exact algorithms, the completeness of the solution is considered, whereas for the approximate algorithms the performance aspects are preferred over exactness.
3. *Output dimension*: This dimension considers different possible forms of outputs generated by matching algorithms. For example, one algorithm can determine only 1-to-1 matches, while another one can also identify 1-to-many matches. Another example is that the results of some types of algorithms are values in the range [0,1] for element pairs being compared, whereas some other types of algorithms identify the match results using some relationship operations, such as 'equivalent'.

Schema matching process may take a variety of inputs and may produce some outputs depending on the matching approach that it applies. Figure 2.7 shows briefly the inputs and outputs for the matching process introduced in this research work, as later explained in this thesis. The variety of inputs consist of the schema specification, a linguistic dictionary, a number of linguistic and structural similarity algorithms, and the user input for validating the results. Output of the matching process is a set of similarity scores for each match identified for schema elements as well as the relationship operations for complex matches, such as string concatenation.

Extensive research has been done in the past in relation to the schema matching field. A number of approaches have been proposed, requiring different amounts of manual intervention from user. More detailed information about these schema matching approaches is given in Chapter 4.

A number of other terms and concepts related to schema matching process have been introduced in the research literature, such as: the ontology matching and mapping discovery. Especially, the ontology matching has drawn considerable attention in recent years with the increasing popularity of the Semantic Web. Although ontology and database schemas have different purposes, the spectrum of "ontology specification" is very broad and a database schema can be considered as a simple descriptive ontology of an environment. In general, "Ontology" is assumed with different meanings and details depending on where it is used. For

example, (Gruber, 1993) defines ontology as a specification of a conceptualization. In this direction, it can be related to a database schema, which in general presents the meta-data defined for a database containing information about the structure and content of that database.

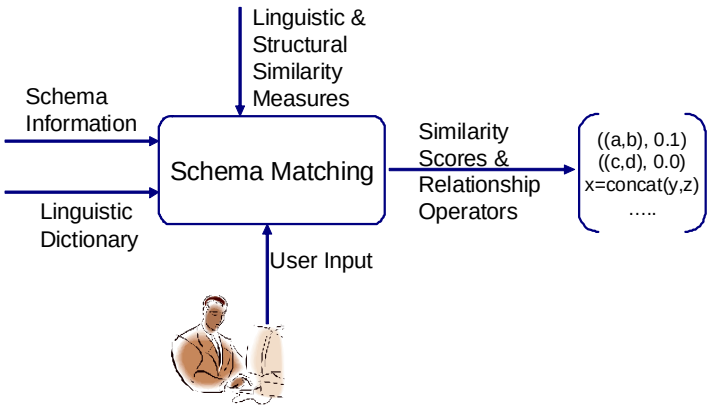


Fig. 2.7. Inputs and Outputs of Schema Matching

Applications of Schema Matching

In addition to its role in the semi-automatic schema integration, the matching process plays an important role in several other application domains, such as in data warehouses, query processing, Semantic Web, and e-business (Rahm & Bernstein, 2001) (Do et al., 2002). Each of these applications needs to deal with some heterogeneous schemas and identify the matches and mappings either manually or semi-automatically. The following paragraphs briefly address these application domains and their relation to matching process.

Data Warehouses

The number of data warehousing applications has increased rapidly in the last decade. Data warehousing has become popular with the need for analyzing large amount of data using different techniques and algorithms to extract information related to a variety of domains, such as sales. Data warehouses aim to most optimally support the analysis and reporting of collected data. In order to form a data warehouse, data from different sources need to be transformed into a common warehouse format. Schema matching process can help in creating an appropriate interlinking and transformation.

E-business and E-commerce

Another application area for schema matching is related to the heavy use of the e-commerce and e-business for transactions among companies. In recent years, these have become popular among both national and international trading partners, using the opportunities that the Internet provides. Using these technologies, partners exchange messages, receive product information,

place orders, sign contracts, etc. Each organization may use different tools and thus different formats for exchanging messages and conducting their transactions, such as the XML, the Electronic Data Interchange (EDI), etc. In order to exchange messages, they need to be translated from one format into another, for which the schema matching process can be applied.

Query Processing on the Web

With the increasing number of data sources available, query processing on the Web has become important. Users pose queries applying their own terminology and the query processing systems need to re-write these queries. For this purpose, the query processing system needs to identify correspondences and mappings between the terms in these queries and the actual terms introduced in the underlying schemas. This is therefore another potential application where schema matching can be utilized.

Semantic Web

The Semantic Web mechanisms contribute to semantically enriching the contents of the Web pages. Semantic enrichment is mainly achieved by associating the concepts on Web pages to ontologies. In other words, the contents of the Web pages are annotated by definitions within these ontologies. However, it is not necessary (and not practiced) that all Web pages use the same ontology. Therefore, before integrating information from different sites, Semantic Web needs to first identify correspondences among different related ontologies that these sites use to annotate their concepts. This is therefore another example of the need for schema matching process, where a semi-automatic schema matching approach can play an important role in matching different ontology elements for Semantic Web applications.

2.4 Conclusion

There are different definitions introduced in the literature for the terms and concepts related to data sharing among distributed nodes. For instance, the concept of ‘Federated Database Architecture’ is interpreted differently by different researchers and authors. This chapter provides some background on the concepts and definitions used by the past research, as related to the subject of this thesis.

Furthermore, for the purposes of setting the context for the problem space addressed in the thesis, and achieving common understanding of the terminology pertaining to the problem area that we try to tackle, this chapter provides our approach on integrated information management system taxonomy. Schema integration and schema matching are two important processes required by the integrated information management systems. As explained in this chapter, these processes need to be automated to the extent possible in order to facilitate easy construction of such information management systems.

Chapter 3

Heterogeneity

Heterogeneity corresponds to differences in a wide range of areas related to information systems, and is considered as the most challenging obstacle standing in the way of achieving seamless interoperability among independent information systems. This chapter first provides some introductory information about three dimensions, distribution, autonomy, and heterogeneity, under which information systems are categorized in relation to accessing information. Then Section 3.2 presents a number of taxonomies for heterogeneities proposed in the literature. Section 3.3 focuses on the main approaches for dealing with the schema heterogeneity, since this constitutes one of the main subjects of this thesis. Also provided under Section 3.3, different types of schema heterogeneities are exemplified. Finally, Section 3.4 concludes this chapter and emphasizes the importance of tackling schema heterogeneity problems with a semi-automated approach.

The research results presented in this chapter were partially published in the Journal of Knowledge and Information Systems (Unal & Afsarmanesh, 2010) and in Lecture Notes in Computer Science (Unal & Afsarmanesh, 2006b).

3.1 Related Concepts

From the viewpoint of accessing the information, the existing information systems are categorized under three dimensions, including: distribution, autonomy, and heterogeneity (Sheth & Larson, 1990), as further detailed below:

- *Distribution Dimension*: Data is typically distributed over different, usually geographically dispersed data sources. With the advances of the Web, these sources are now interlinked, hiding their physical locations, and thus making this dimension less challenging with regards to achieving database interoperability. However, exchanging large volumes of data over distributed networks have been another challenging aspect, which can now be easily dealt with through broad bandwidths.
- *Autonomy Dimension:* Each organization runs some information systems independently from others. For example, organizations may autonomously decide to share a part of their local resources or services with others. Furthermore, they may maintain autonomy on their local data and define/use their own data models. We identify four main types of autonomy that can be exercised in federated database systems: 1) *Design autonomy* that refers to a component's being independent from others in their information system design, including data model, query language, constraints, etc. 2) *Communication autonomy* that refers to a component's autonomy in deciding whether or not to communicate with others and when and how to communicate. 3) *Association autonomy* that refers to a component's autonomy in deciding which parts of its resources and functionalities to share with others.

4) *Execution autonomy* that enables a component to execute local operations and to decide on the order of these operations without interference of external systems.

- *Heterogeneity Dimension:* Heterogeneity arises due to autonomy of organizations. It corresponds to differences in numerous areas of information systems. Heterogeneity has been the most challenging dimension within the context of database interoperability, especially considering the large variety of conflicts that may exist among distinct data providers. Different types of classifications are proposed in the literature for this dimension, as addressed in the next section.

3.2 Taxonomy of Heterogeneity Resulted Conflicts

Different, but partially overlapping classifications for heterogeneity have been proposed in the literature. Some classifications only distinguish between the information and schema when it comes to heterogeneity, while some others consider several other types of heterogeneity in information systems. In this section, we present five different classifications defined in the literature, in relation to our research work:

Classification-1: As for conflicts that may exist among schemas, Batini et al. (Batini et al., 1986) defines two categories, as shown below in Figure 3.1, including: *name conflicts* and *structure conflicts*.

1- ***Name conflicts*** arise because of the fact that different database designers typically use different terminology for the same domain. Typically, there are two types of relationships among the names used that cause name conflicts:

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

2- ***Structural conflicts*** arise because of using either different modeling constructs or different integrity constraints. Structure conflicts are classified by (Batini et al., 1986) as follows:

- **Type Conflicts** occur as a result of using different modeling constructs (e.g. using entity versus attribute) for representing the same concept.
- **Dependency Conflicts** arise when different schemas introduce different relationships among the same concepts, such as a 1-to-1 relationship is indicated between two concepts in one schema, while in another schema the concepts have a 1-to-m relationship.

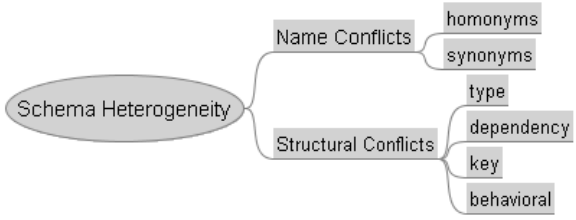


Fig. 3.1. Taxonomy of Schema Conflicts (Batini et al., 1986)

- **Key Conflicts** arise when different keys are introduced for the same concept in different schemas. For example, *employee* may have the *SSN* attribute as the key in one schema, whereas in another schema its key may be the *ID* attribute.
- **Behavioral Conflicts** arise due to different insertion / deletion policies introduced in different schemas for the same concept. For example, in one schema *player* data may exist without a related *team* data, but in another schema when a *player* is inserted it has to have a related *team* data.

Classification-2: In another study, (Sheth & Kashyap, 1992) defines a classification with the emphasis on the schematic heterogeneities and semantic similarities, as shown in Figure 3.2.

1- ***Domain Definition Incompatibility*** corresponds to differences in attribute domain definitions for representing semantically similar attributes and consists of types of conflicts listed below:

- **Naming Conflicts**: Using synonyms and homonyms for defining attributes.
- **Data Representation Conflicts**: Using different data types for semantically similar attributes. For example, an attribute may be defined of type *string* in one schema, whereas in another schema a similar attribute may be defined of type *integer*.
- **Data Scaling Conflicts**: Using different units or measures. For example, *price* attribute might have values in *dollar* in one database and in *euro* in another database.
- **Data Precision Conflicts**: Using different precisions. For example, the *grade* attribute may have a value between 1-100 in one schema, but it may have a letter value (A, B, C, etc.) in another schema (Sheth & Kashyap, 1992).
- **Default Value Conflicts**: Using different default values. For example, default value for the threshold attribute might be 0.5 in one schema and 0.6 in another schema.
- **Attribute Integrity Constraint Conflicts**: Using different integrity constraints that might not be consistent with each other. For example, an attribute *X* may have the constraint that *X>30* in one schema and *X<20* in another schema.

2- ***Data Value Incompatibility*** corresponds to using different values for data in different databases. This type of incompatibility depends on the database state and can arise as a result of the following inconsistencies:

- **Known Inconsistency**: Related to inconsistencies, cause of which are known. For example, it might be known that one database is more reliable than the other. In this case, the more reliable database can be used to resolve the inconsistency.
- **Temporal Inconsistency**: Related to inconsistencies, which are temporal. In other words, information stored in a database is time dependent. In this case, inconsistency is temporary.
- **Acceptable Inconsistency**: Related to inconsistencies that are within an acceptable range and considered tolerable. Therefore, for some types of queries, the errors in the values of inconsistent databases may be tolerable.

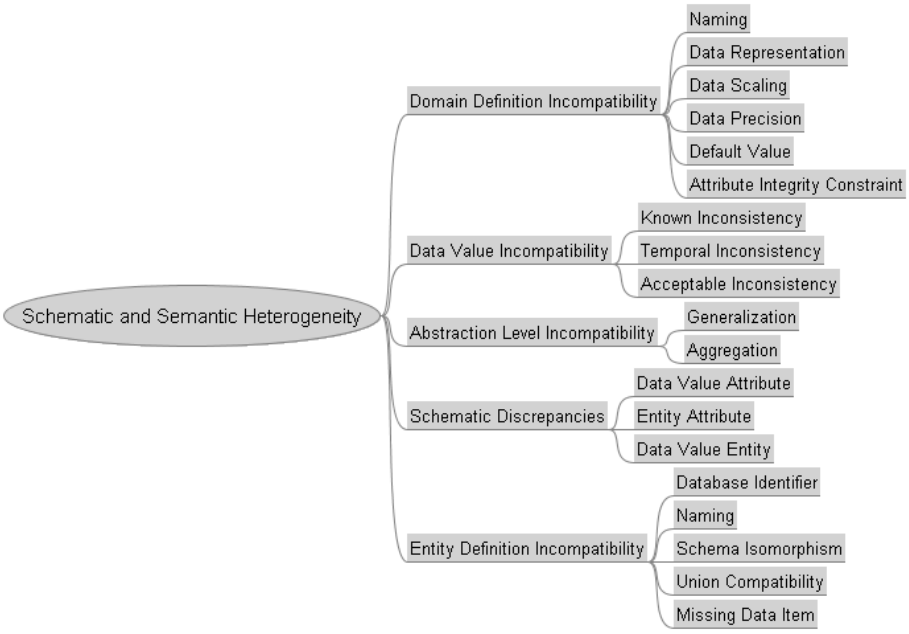


Fig. 3.2. Taxonomy of Schema Conflicts (Sheth & Kashyap, 1992)

3- ***Abstraction Level Incompatibility*** corresponds to differences in levels of abstraction that different databases use to represent semantically similar entities and can be of two types:

- **Generalization Conflicts**: Using different levels of generalization. For example, *car* entity may be represented by the *vehicle* entity in one schema and *car* entity in another schema.
- **Aggregation Conflicts**: Using aggregation in one database to identify a set of entities in another database. For example, *team* in one schema is a set of *player*s in another schema.

4- ***Schematic Discrepancies*** arise when data in one database corresponds to metadata in another database and consist of three types of conflicts:

- **Data Value Attribute Conflict**: Arises when the value of an attribute in one database corresponds to an attribute in another database. For example, letter grades (A, B, C, D, F) of a student may be stored in a *grade* attribute in schema S1, whereas in *A*, *B*, *C*, *D*, and *F* attributes in schema S2.
- **Entity Attribute Conflict**: Arises when an entity is modeled as an attribute in one database, whereas as a relation in another database. Considering the example in the Data Value Attribute Conflict, suppose that another schema, S3, has separate relations for each grade, in the form of A(date, name_of_student,..), B(date, name_of_student,..), etc. In this case, there is an entity-attribute conflict between the schemas S3 and S1.

- **Data Value Entity Conflict**: Arises when the value of an entity in one database corresponds to a relation in another database. Considering the examples in data value attribute and entity attribute conflicts, there is a data value entity conflict between the schemas S2 and S3.

5- ***Entity Definition Incompatibility*** arises when incompatible entity descriptors are used and involves the following types:

- **Database Identifier Conflicts**: Using semantically different identifier records. For example, *employee* entity may have the attribute *ssn* as the key in one schema and *name* as the key in another schema.
- **Naming Conflicts**: Using synonyms and homonyms for defining entities. Name conflicts here exist among entities, whereas the name conflicts under "Domain Definition Incompatibility" exist among attributes.
- **Schema Isomorphism Conflicts**: Using different number of attributes for semantically similar entities.
- **Union Compatibility Conflicts**: Having semantically unrelated set of attributes for semantically similar entities. For example, *employee* entity having *ssn, name,* and *address* attributes in one schema and *ssn, name,* and *salary* attributes in another schema are union incompatible.
- **Missing Data Item Conflicts**: Arise when one of the semantically similar elements has a missing attribute. For example, *vehicle* entity may have *type* attribute (for the types of vehicle, such as car, bus, etc.) in one schema, but in another schema, *car* entity may not have this attribute.

Classification-3: Another classification, shown in Figure 3.3, is proposed by Sheth for defining different types of heterogeneity in information systems (Sheth, 1998):

1- ***Information Heterogeneity:*** Corresponds to differences in information that involves *semantic*, *structural* and *schematic*, and *syntactic* heterogeneities.

2- ***System Heterogeneity:*** Corresponds to differences in information systems, namely in *digital media repository management systems* and *database management systems*

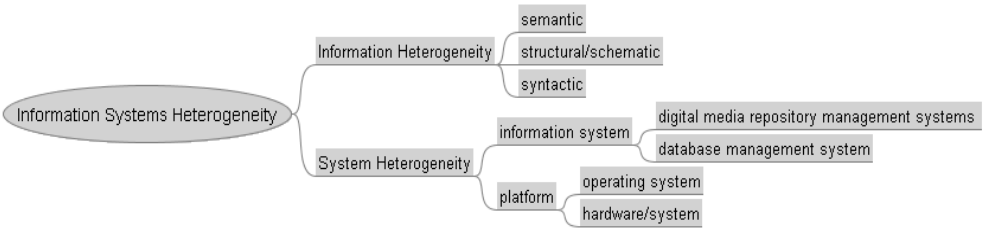


Fig. 3.3. Taxonomy of Information Systems Heterogeneity (Sheth, 1998)

(DBMS), as well as disparities in *platforms*, namely *operating systems* and *hardware/systems*.

Classification-4: (Busse et al., 1999) uses another classification, shown in Figure 3.4, which divides the heterogeneity into three: syntactical, data model and logical heterogeneity, as explained below.

1. ***Syntactical Heterogeneity*** is divided into two main types; technical and interface heterogeneity:
   - **Technical Heterogeneity**: Differences in technical aspects such as operating systems, protocols, etc.
   - **Interface Heterogeneity**: Heterogeneity that arises as a result of using different access languages. Differences in the following subjects cause interface heterogeneity:
     - *Language Heterogeneity*: Use of different query languages or language restrictions.
     - *Query Restrictions*: Allowing only certain types of queries, such as the maximum number of joins allowed.
     - *Binding Restrictions*: Allowing only certain attribute values in queries.
2. ***Data Model Heterogeneity***: Related to different data models' having different semantics for their concepts.
3. ***Logical Heterogeneity***: Classified in three sub-categories:
   - **Semantic Heterogeneity**: Differences in semantics of data and schema. Different semantic schema conflicts can occur: equal names can denote different concepts (homonyms), different names can be used for the same concept (synonyms), and so on.
   - **Schematic Heterogeneity**: Differences in encoding of concepts at different elements of a data model. Attribute name-relationship and attribute name-attribute value conflicts in relational databases are examples of this kind of heterogeneity.
   - **Structural Heterogeneity**: Occurs if elements are structured in different ways in

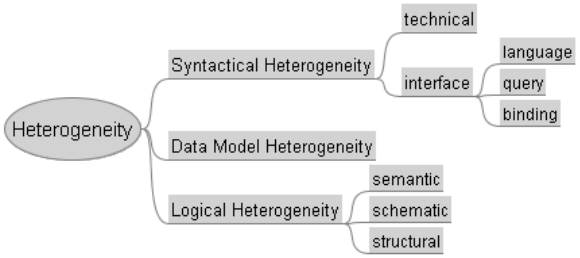


Fig. 3.4. Taxonomy of Information Heterogeneity (Busse et al., 1999)

different schemas. For example, grouping attributes into different tables in two schemas results in this type of heterogeneity.

Classification-5: Another classification of heterogeneity is proposed by (Kahng & Mcleod, 2001), who divides information heterogeneity into two, as depicted in Figure 3.5:

1. ***Data Model Heterogeneity:*** Differences in collections of structures, constraints, and operations that information systems use to describe and manipulate data.
2. ***Semantic Heterogeneity:*** Differences in specifications of data. Semantic conflicts among information systems that use object-based data model are listed by (Kahng & Mcleod, 2001) as follows:
    - **Category***:* Two objects coming from different information sources may have equivalence, sub-concept/super-concept, or partially overlapping relationships when they represent same or similar real world entities.
    - **Structure***:* Two objects coming from compatible categories may have different structures. For example one object in one system may have one attribute but another object in another system might not have it.
    - **Unit**: Two objects coming from the compatible categories and having the same structures may use different units.
    - **Terminology**: Use of synonyms and homonyms may cause semantic heterogeneities.
    - **Universe of Discourse**: Semantics hidden in the context may result in semantic heterogeneities.

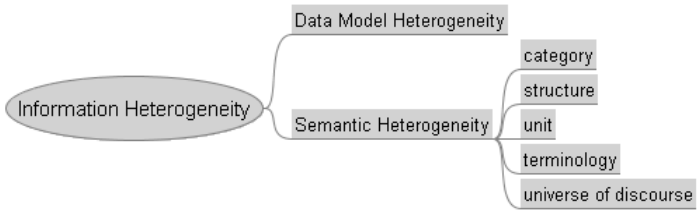


Fig. 3.5. Taxonomy of Information Heterogeneity (Kahng & Mcleod, 2001)

3.3 Challenges for Schema Matching

As it is clear from the existence of a large number of classifications presented in Section 3.2 above, heterogeneity has been one fundamental problem against enabling interoperability among information systems. Despite the existence of different classifications of heterogeneity, there are many commonalities in the terminology that these classifications employ. However, categories and the terms used in each classification are confusing. For example, Classification-

2 places “Naming Conflicts” under both “Domain Definition Incompatibility” and “Entity Definition Incompatibility”. The first one refers to the attributes, whereas the second one refers to the entities. Furthermore, for the “Schematic Discrepancies” category, it is difficult to infer by just looking at the name to which the category is related for its data-metadata conflicts. As another example, Classification-4 has a category called “Schematic Heterogeneity”, but another category called “Semantic Heterogeneity” is also defined. But there is some overlap between these two categories. Furthermore, some classifications miss to consider heterogeneities corresponding to the different forms of names used in the schemas, such as using abbreviated vs. extended names.

Since the focus of this thesis is on schema matching and schema integration, we concentrate on analyzing schema heterogeneities. Hence, the heterogeneities related to instance data and underlying systems, such as those of the database management system are not considered in our classification. Furthermore, we aim to define a classification that is clear and simple (as opposed to those other classifications that are confusing), but at the same time broad enough to cover a wide variety of different types of database schema conflicts. In this respect, the research explained in this thesis covers both the structural and linguistic heterogeneities, to capture the semantic, syntactic, and structural schema conflicts, as indicated in Figure 3.6.

A number of examples are provided below, each of which representing a different kind of schema conflict that belongs to one of the categories: structural, linguistic, or belong to a combination of the two. These examples are taken from two specific example schemas S1 and S2, defined for a university domain, encompassing courses, students, etc., as shown in Figure 3.7. As described in these examples, structural conflicts are more difficult to resolve than linguistic conflicts. Clearly, the varieties of types of conflicts that exist among databases are not limited to these given here. However, the conflicts shown in the examples are the ones which frequently occur in database schemas, and cause bottlenecks for automatic schema matching and integration. Note that for simplicity reasons, in the example cases only partial schemas are shown in a simple format. If relevant to the example, some primary and foreign keys are also shown in the schemas.

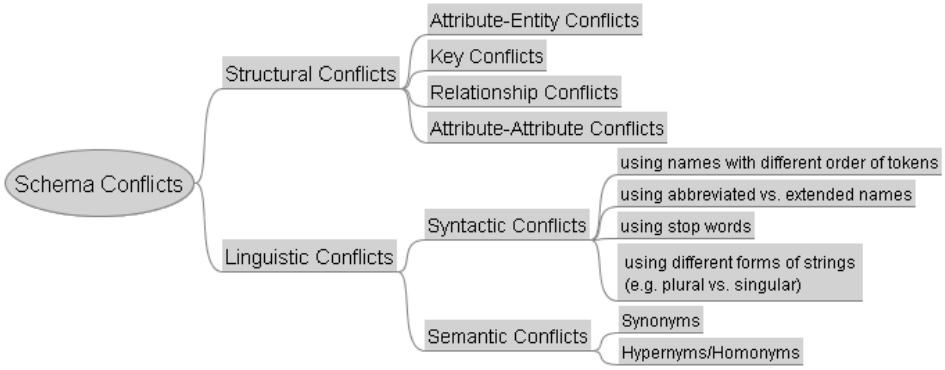


Fig. 3.6. Classification of Schema Conflicts Considered in the Thesis

S1
Person (ssn (PK), fname, lname, rank, salary, office, phone, child, depid (FK))
Student (stid (PK), fname, lname, birth_date, sex, address, class, gpa, depid (FK))
GradStudent (stid (PK, FK))
Department (dptID (PK), name_of_department, dphone, doffice, college)
Course (cno, cname, description, section, day, ssn (FK), depid (FK))
Time (timeid, start, end, cno(FK))
Campus (cid (FK), location)

S2
Instructor (id (PK), name, telephone, children, dcode (FK))
Student (studentID (PK), degree, date_birth, name, gender, city, state, zip,
GradePointAverage, dptcode (FK))
GradeReport (studentID (PK, FK), section_number (FK), letter_grade, numeric_grade)
Course (cnumber, coursename, coursedesc, credits, dcode (FK))
Section (section_number (PK), semester, year, instID (FK), day, cnumber (FK))
Time (timeid, start, end, section_number (FK))
Department (dptCode (PK), nameDepartment, officeNumber, officePhone, college)
Campus (cid (PK), city, zipcode)
Apply (studentID (PK, FK), cid (PK, FK))

Fig. 3.7. Example Schemas from University Domain

1) ***Examples of Structural Conflicts:*** Structural conflicts exist due to the fact that different organizations use different constructs and integrity constraints to represent the same concepts.

- **Attribute-Entity Conflict***:* This conflict arises when a concept is represented as an attribute in one schema and as a separate entity in the second schema. For example, as shown in Figure 3.8, "section" information is represented by the 'section' column of the "Course" table in schema S1, while schema S2 represents it as a separate table "Section".

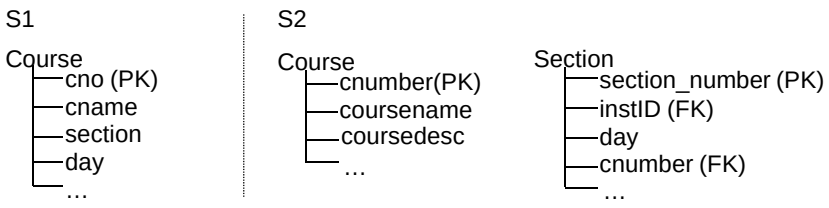


Fig. 3.8. Attribute-Entity conflict

- **Key Conflict:** This conflict arises when different keys are assigned for the same entity in different schemas. In the example shown in Figure 3.9, the key of the "Department" table in schema S1 is "dptID", while that of the "Department" table in schema S2 is "dptCode"

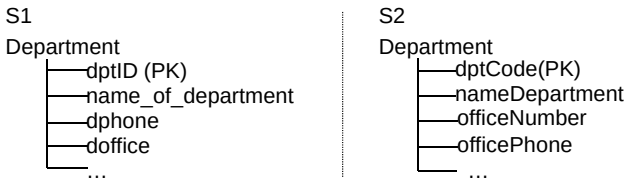


Fig. 3.9. Key conflict

- **Relationship Conflict:** This conflict is related to the case, in which relationships between entities in different schemas are defined differently, as exemplified in Figure 3.10. In S1, there is a one-to-many relationship between the "Student" and "Campus" tables, while in S2, this relationship is many-to-many, thus introducing a third table "Apply", to represent this relationship.

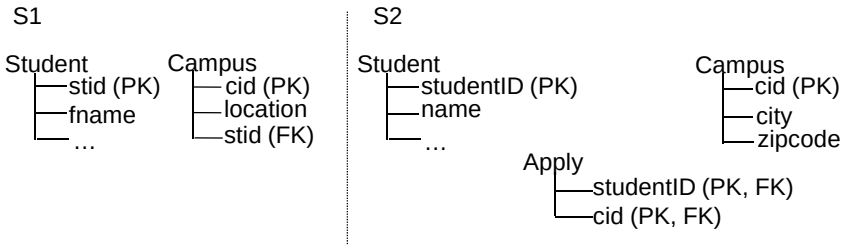


Fig. 3.10. Relationship conflict

- **Attribute-Attribute Conflict**: This conflict arises when the same concept is represented by using different number of attributes in different schemas. For the example shown in Figure 3.11, "address" data is stored in one column of the "Student" table in S1, while in S2, it is spread over three columns.

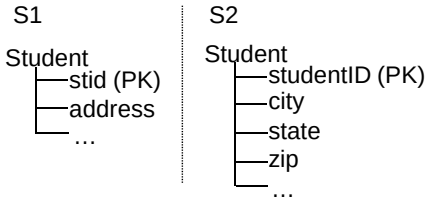


Fig. 3.11. Attribute-Attribute conflict

2) ***Examples of Linguistic Conflicts*****:** Linguistic conflicts arise because of different terminology and names that different organizations use to refer to the same data. Linguistic conflicts can be of two types: syntactic and semantic. Below are the examples for the syntactic

and semantic conflicts. Since these types of schema conflicts are due to differences in representation of "names" of columns or tables, we have not provided example schemas but only some related but different names in the following examples.

- **Syntactic Conflicts: These conflicts arise** due to using different forms of the names and can be of the following types:
    - Using names that consist of more than 1 token (word) with different order of tokens, e.g. *birth_date* vs. *date_birth*.
    - Using abbreviated vs. extended names, e.g. *GPA* vs. *GradePointAverage.*
    - Existence of stop words, e.g. *name_of_department vs. nameDepartment*
    - Using different forms of a string (plural, singular, etc.), e.g. *child* vs. *children.*
    - Similar to the use of abbreviated vs. extended names, use of short forms of strings, e.g. *phone* vs. *telephone.*
- **Semantic Conflicts: These conflicts arise** due to differences in semantics of names and can be of the following types:
    - Use of synonyms, e.g. *gender* vs. *sex.*
    - Use of hypernymy / hyponymy (representing IS-A), e.g. *person* vs. *instructor*.

3) ***Example of Combined Structural and Linguistic Conflicts*****:** The two types of conflicts addressed above may occur in a combined form in some parts of the schema. Following figure (Figure 3.12) shows an example for this case, where the "location" information in S1 can be matched with the concatenation of "city" and "zipcode" columns of the "Campus" table in S2, if we apply both the linguistic "IS-A hierarchy" and the structural "one attribute in the first schema matches two attributes in the second" (attribute-attribute conflict) conflict resolution techniques.

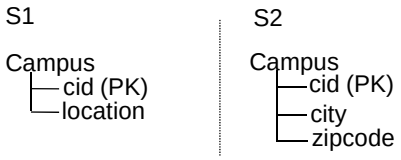


Fig. 3.12. Combined Structural and Linguistic conflicts case

3.4 Conclusion

The term heterogeneity, also referred to as conflict in several related reported research, is used in different contexts, such as those addressing schema heterogeneity, information heterogeneity, etc. It is generally used to refer to the diversity or difference, which exists among distinct elements within a domain. Heterogeneity is a big obstacle to interoperability. In information systems, with the increasing number of efforts during the last years, which aim

to enable interoperability, heterogeneity has been addressed as an important issue common among them.

Different classifications of heterogeneities in information systems have been proposed in the related research literature. Considering the subject of this thesis, schema heterogeneity is the one, on which we have focused. We divide schema heterogeneity into two main classes of: *structural* and *linguistic*. *Structural conflicts* exist due to the fact that different organizations use different constructs and different integrity constraints to represent the same or similar concepts. The most frequently occurring structural conflicts include: *attribute-entity conflicts*, *key conflicts*, *relationship conflicts*, and *attribute-attribute conflicts*. *Linguistic conflicts* on the other hand arise because of the different terminology and names that different organizations use to refer to the same or similar concept. Linguistic conflicts can be of two types: *syntactic* and *semantic*. Structural conflicts are more difficult to identify and resolve than linguistic conflicts.

In any approach that attempts to enable data sharing among heterogeneous databases, both linguistic and structural conflicts need to be considered. Considering that the complete semantics of the concepts introduced in an environment are not and/or cannot be fully expressed in database schemas, human intervention is deemed to be necessary, as the decision maker to resolve ambiguities in this area. Therefore, fully-automated conflict resolution does not seem realistic for the time being, and hence we hypothesize that semi-automatic approaches need to be devised to more effectively deal with the problem of heterogeneity.

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.

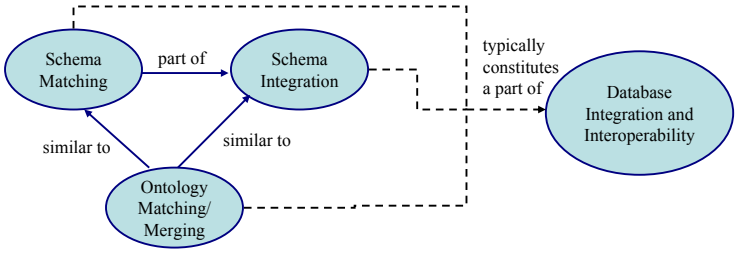


Fig. 4.1. Concept Map of Related Research Approaches

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.

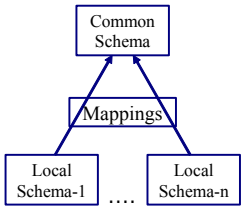


Fig. 4.2-a. Database Federation with common schema

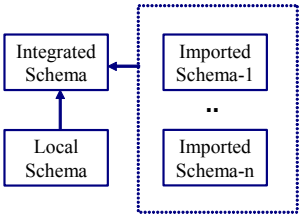


Fig. 4.2-b. Full database federation

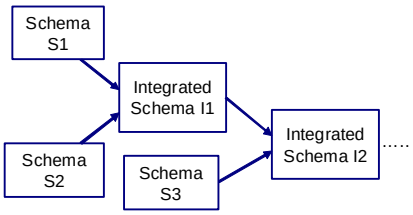


Fig. 4.2-c. Incremental generation of a global integrated schema

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).

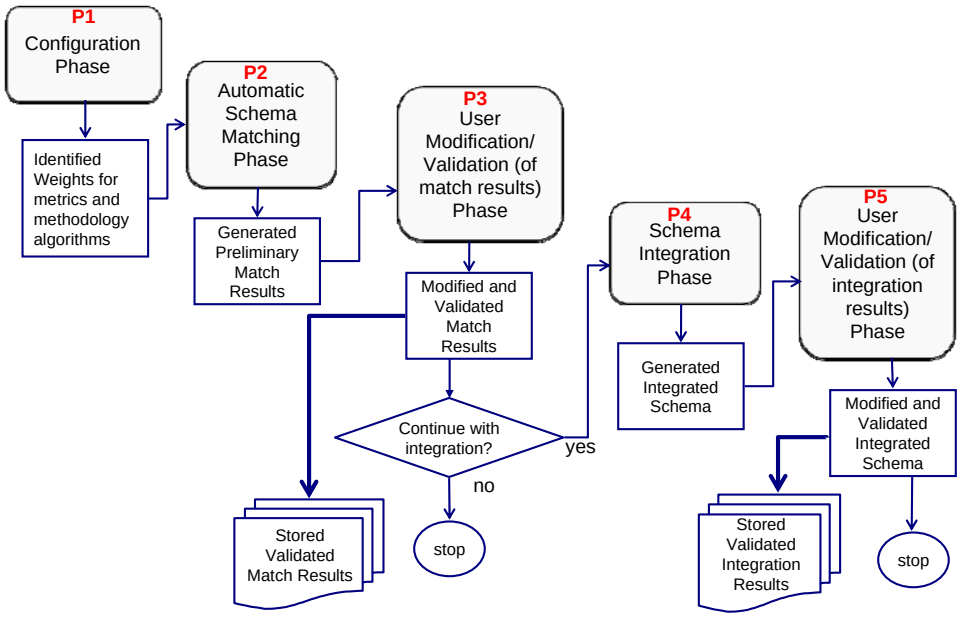


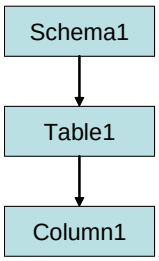
Fig. 4.3. Main Phases of SASMINT

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.

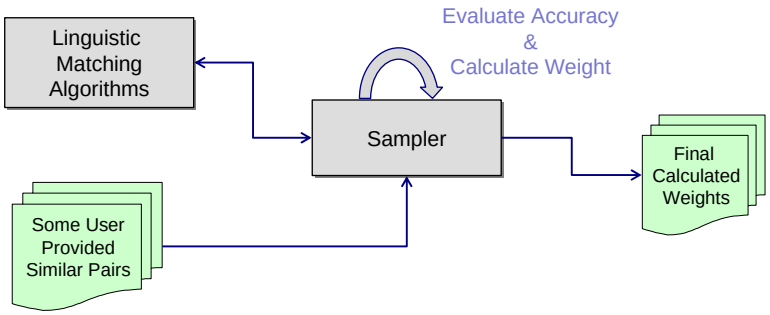


Fig. 4.7 Operation of Sampler

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.

Chapter 5

SASMINT development architecture

We have implemented the SASMINT system to verify, test, and validate the approach proposed in this research as well as to compare it with other approaches. Details of the development architecture of SASMINT are provided in the following sections, together with a number of screenshots of this system. Section 5.1 goes over the processing steps of SASMINT. Section 5.2 briefly addresses the technologies applied in the development of SASMINT. Section 5.3 lists the main components of the system. Detailed explanations about the operation of the SASMINT system are provided in Section 5.4. Finally, Section 5.5 concludes this chapter.

A part of the research results presented in this chapter was previously published in two articles, of which one appeared in the Journal of Knowledge and Information Systems (Unal & Afsarmanesh, 2010), and the other appeared in the Journal of Software (Unal & Afsarmanesh, 2009).

5.1 Processing Steps of SASMINT

SASMINT realizes the approach described in Chapter 4, following the processing steps shown in Figure 5.1. To state it briefly, the configuration step allows the user to first identify the weights for matching algorithms and then identify the strategy for selection of the results of schema matching. Schema matching step starts with the preparation sub-step that translates both recipient and donor schemas into the DAG format. Then, based on the approaches explained in Chapter 4, in the comparison sub-step, the linguistic and structure matching take place. After the matching results are generated by SASMINT, users apply their desired modifications and/or validate the proposed matches before the eventual match results are formally defined and persisted. Using these results, the SASMINT system then starts generating an integrated schema, by applying a set of integration rules defined for relational databases. When the results of schema integration are ready, these are again presented to the user for modification/validation of their integration.

5.2 Technologies Applied

Considering the practical environment of collaborative networks that will need to run SASMINT, this system is implemented in Java programming language on Microsoft Windows operating system environment. Being java-based, SASMINT can actually run on multiple operating platforms/systems; i.e. it runs cross-platform. We have therefore also tested it for

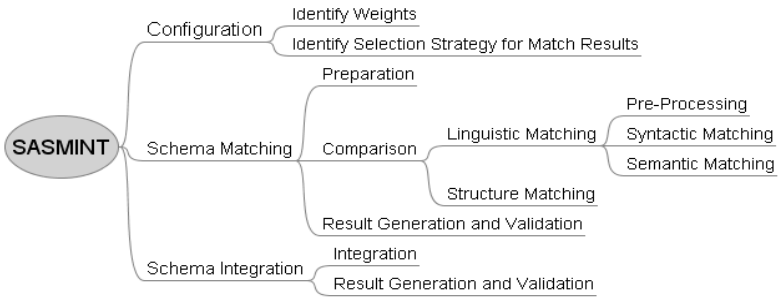


Fig. 5.1. Processing Steps of SASMINT

this purpose and within hours, this system was up and running on MacOSX platform, which actually is a Unix clone from the Operating System point of view. SASMINT is a standalone java swing based application. As far as its architecture is concerned, it has a 2-tiered architecture, where its 'business' tier provides services/functions to its presentation tier. These business layers tiers encapsulate implementations of several business rules.

We have focused our development efforts on supporting the matching and integration of relational schema based systems. The specific technologies and tools exploited in the development of SASMINT are listed below.

- *Eclipse:* Java Integrated Development Environment (IDE). It is a platform for building integrated web and application development tooling (Eclipse, 2010).
- *NetBeans*: Java Integrated Development Environment (IDE). It is a platform for building integrated java applications. We have extensively used NetBeans for the GUI design phases.
- *AWT (Abstract Windowing Toolkit) and Swing:* Java's Graphical User Interface (GUI) libraries.
- *JGraphT:* Free Java graph library to create (model) graphs (Jgrapht, 2010). It supports various types of graphs, such as weighted, unweighted, directed, undirected, and labeled graphs.
- *JGraph:* Graph component for visualization and layout (Jgraph, 2010). Graphs generated using JGraphT can be visualized and the layout can be applied by means of JGraph.
- *WordNet:* A lexical dictionary (Fellbaum, 1998; Wordnet). Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by a number of semantic relationships, such as hypernymy and hyponymy.
- *JWNL (Java WordNet Library)*: A Java API for accessing WordNet (Jwnl, 2010).
- *XML-Beans:* Technology for accessing XML by binding it to Java types (Xmlbeans, 2010). XMLBeans uses XML Schema to compile Java interfaces and classes that can be used to access and modify XML instance data.
- *SecondString:* An open source Java package consisting of implementation of a number of string similarity metrics (Secondstring, 2010). For the purpose of

syntactic matching in SASMINT we modified this package for implementing the Levenshtein, Monge-Elkan, Jaro, TF*IDF, and Jaccard metrics.

5.3 Main Components of the System

The main components of the SASMINT system are illustrated in Figure 5.2. The *Sampler Component* helps users identify appropriate weight for each algorithm used in the linguistic matching. The *Graph Representation Component* of SASMINT is responsible for representing schemas in Graph format, more specifically in the DAG format. SASMINT uses JGraph for graphical representation (visualization) and specifying the layout of the graphs. This system utilizes the Java graph libraries of JGraphT, for creating a graph and performing some operations on the graph, such as getting all its vertices and edges and traversing the graph. Users interact with the system using the *GUI Component*. After the schemas, represented as graphs, have been displayed using the GUI and the user has selected the Match option, the GUI component calls the *Schema Matching Component.* This component matches source and target schemas using a combination of Linguistic and Structure Matching techniques, as explained in Chapter 4. After modifying and validating the match results, the user may continue with schema integration. The *Schema Integration Component* integrates the schemas using a number of pre-defined rules and represents the automatically generated integrated schema to the user, while also formalizing it in a derivation language. Results of integration will go again through the user validation stage.

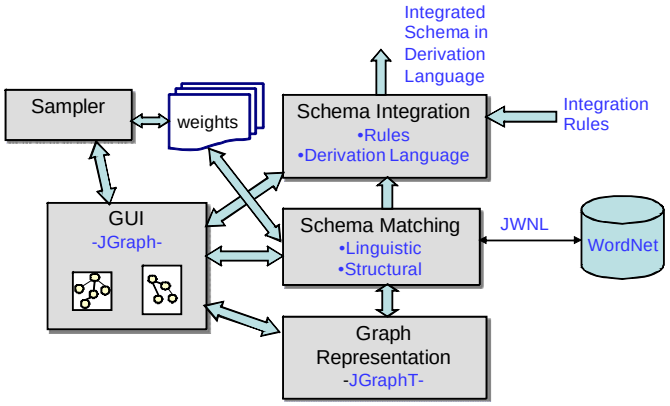


Fig. 5.2. Components of SASMINT

5.4 How does the System Work?

The main flow of information in the SASMINT system as well as the technologies used in different stages is given in Figure 5.3. Firstly, weight for each schema matching algorithm is assigned and the strategy for selecting the match results is identified and threshold value is set, as explained in details in Section 5.4.1. Secondly, the two schemas, called as recipient and

donor, are translated into the graph format and then loaded into the system to be visualized by the SASMINT GUI. Thirdly, correspondences between these two schemas are identified. Finally, the match results are presented to the user, and after modifying and/or validating the match results, if the user continues with the schema integration according to specified matches, these two schemas are integrated into a single schema and then it will be formalized in the SDML, which again requires user validation.

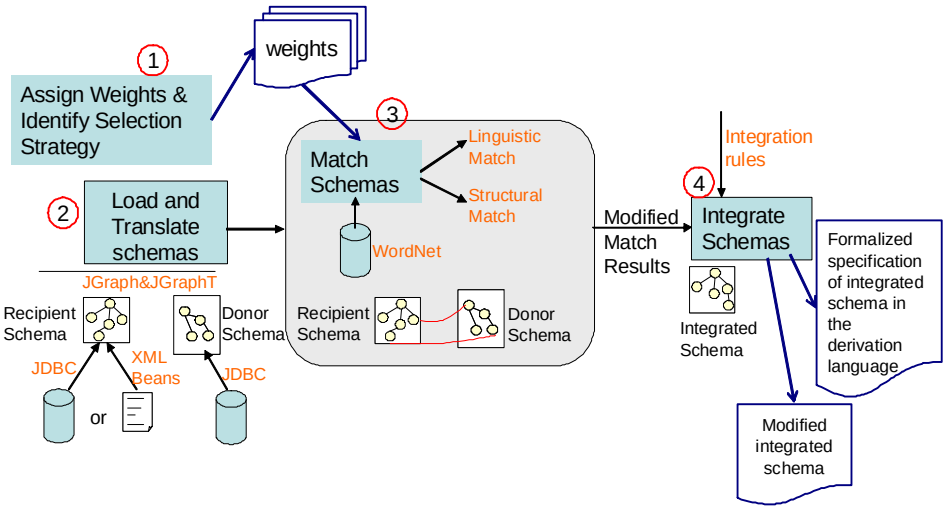


Fig. 5.3. Information Flow in SASMINT

In the following sections, some more details of the operation of the SASMINT system are provided. For this purpose example schemas shown in Figures 5.4-a and 5.4-b are used. Through the screenshots, also the matching and integration of recipient and donor schemas are explained.

```
CREATE TABLE `employee` (
  `empid` int(10) unsigned NOT NULL
auto_increment,
  `fname` varchar(45) NOT NULL,
  `lname` varchar(45) NOT NULL,
  `address` varchar(45) NOT NULL,
  PRIMARY KEY (`empid`)
)
```

Fig. 5.4-a. Recipient Schema

```
CREATE TABLE `address` (
  `addrid` int(10) unsigned NOT NULL auto_increment,
  `street` varchar(45) NOT NULL,
  `zip` varchar(45) NOT NULL,
  `city` varchar(45) NOT NULL,
  PRIMARY KEY (`addrid`)
)
CREATE TABLE `person` (
  `pid` int(10) unsigned NOT NULL auto_increment,
  `name` varchar(45) NOT NULL,
  `birth` varchar(45) NOT NULL,
  `addressid` int(10) unsigned NOT NULL,
  PRIMARY KEY (`pid`),
  KEY `FK_person_1` (`addressid`),
  CONSTRAINT `FK_person_1` FOREIGN KEY (`addressid`) REFERENCES
`address` (`addrid`)
)
```

Fig. 5.4-b. Donor Schema

5.4.1 Assigning Weights and Identifying the Selection Strategy

In the three methods introduced for the configuration phase of SASMINT in Chapter 4 (Section 4.2.2), it is stated that in one method, the weights for each algorithm used for schema matching can be manually assigned, as shown in Figure 5.5, while in another method, default weights are assigned by the SASMINT system. The default weight for each algorithm is the equal weight in the related category of matching. For example, the default weight for the six considered metrics in the syntactic similarity is 1.0 / 6, which is ≈0.16. Furthermore, for combining the results of linguistic and structure matching for the final result, default weight of linguistic matching is 0.7 and that of structure matching is 0.3. As explained in Section 4.2.3.3, these weights are also modifiable through the GUI, shown in Figure 5.5.

Since it may be difficult for an average user to identify appropriate weights, SASMINT introduces and supports a third method implemented by the Sampler component for automatic weight identification, indicating which algorithm/algorithms better suited for the specific domain of the schemas. Sampler can be applied to the calculation of both syntactic and semantic similarity metrics.

With the current implementation, the Sampler component can work with up to five known sample pairs (this is merely a design feature, and is thus easily customizable). Through the GUI provided by the Sampler component, as shown in Figure 5.6, the user has the freedom to provide: a) syntactically similar pairs in case he/she would like the system to compute the weights of syntactic matching metrics, or b) semantically similar pairs in case he/she requires to compute the weights of metrics for semantic matching. The user is expected to input these pairs to the Sampler component from his/her schema domain. For instance, the user might want to see how syntactic similarity metrics would perform for the pair P: ["student_name", "name_of_student"]. On the other hand, he might want to see how semantic similarity metrics would perform for the pair P: [“employee”, “worker”]. After doing some computations, as explained in Chapter 4, the Sampler component outputs the appropriate weights for the schemas’ domain, as can be seen in Figure 5.6.

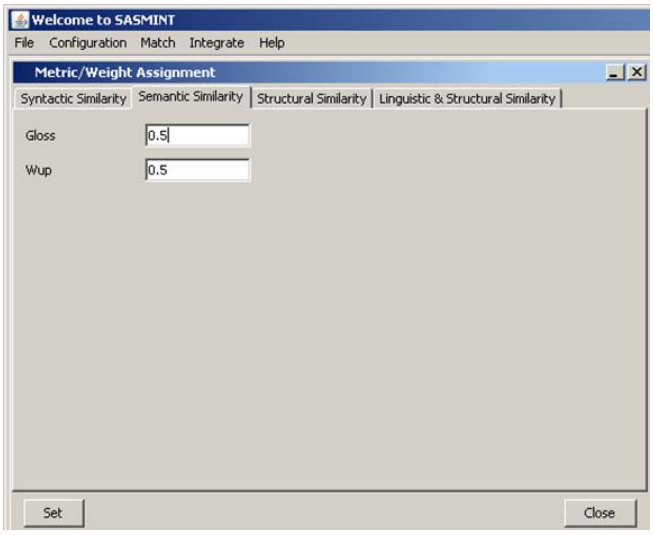


Fig. 5.5. Manual Weight Assignment

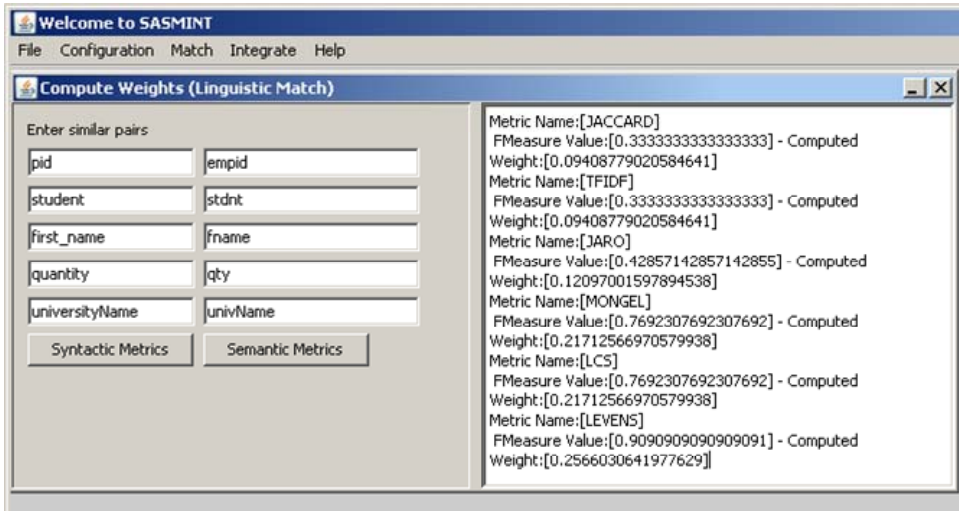


Fig. 5.6. Use of Sampler

For identifying the selection criteria for the results of schema matching, user is required to set a threshold value as well as the strategy to use for selecting the results of schema matching. Figure 5.7 shows the screenshot relevant to this example from SASMINT. The meanings of 'select all above threshold' and 'select max above threshold' are explained in Chapter 4 (Section 4.2.2). If nothing is specified by the user, default value for threshold is 0.5 and default strategy is "select max above threshold".

Fig. 5.7. Identifying Result Selection Strategy

5.4.2 Loading and Translating Schemas

After assigning weights to the algorithms and identifying the selection strategy (e.g. 'select all above threshold' and 'select max above threshold'), two schemas, called the *recipient* and *donor* respectively, are loaded by the user into SASMINT, through its GUI, as shown in Figure 5.8. The recipient schema is taken as the base schema in the integration process. It can be loaded either from a database or from an XML file, which contains a previously generated integrated schema and all its related derivation information. Format of this file is described in the following paragraphs about the SDML. It is assumed that XML file contains an SDML-based representation of the integrated schema, which is loaded as the recipient schema. On the other hand, donor schema can be loaded from any database. During the loading process, schemas specified in the language of their database are translated into a Directed Acyclic Graph (DAG) format. In other words, table and column names as well as the primary and foreign keys will all be represented in graphs for the recipient and donor schemas, before their processing starts. When a recipient schema is loaded from an XML file, only this information is shown in the graph, not its derivation information. DAG is used as the common format for representing schemas. SASMINT uses JGraphT, a free Java graph library (Jgrapht, 2010), to create the DAG.

When the two graphs, corresponding to donor and recipient schemas are generated, they are displayed for user through the SASMINT GUI. A graph component, called JGraph and its subcomponent, JGraph Layout are used for graph visualization and layout (Jgraph, 2010). The processes of SASMINT, responsible for loading schemas and translating them into the graph format constitute the *Preparation* sub-step.

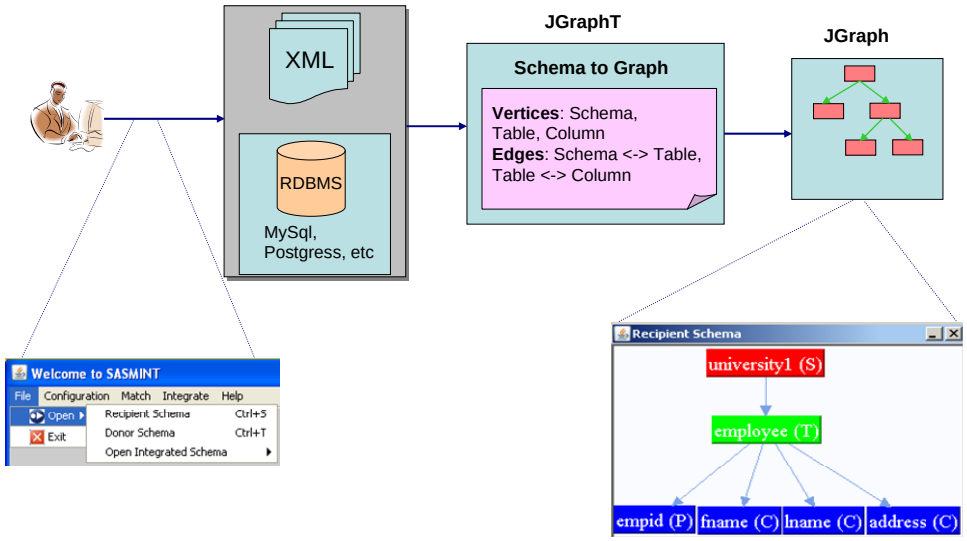


Fig. 5.8. Preparation Sub-step

Figure 5.9 illustrates the screenshots of the two schemas given in Figure 5.4 in the MySQL database. Figure 5.10 shows how they are represented as graphs when loaded into SASMINT. Schemas, tables, and columns are shown in different colors to help users during the modification and validation of the schema matching results. Furthermore, primary and foreign key columns are also indicated in the views, by appending a "(P)" for a primary key and a "(F)" for a foreign key in the node name. JGraph provides a flexible visualization for graphs. It allows users to easily move the nodes in order to modify the appearance of a graph.

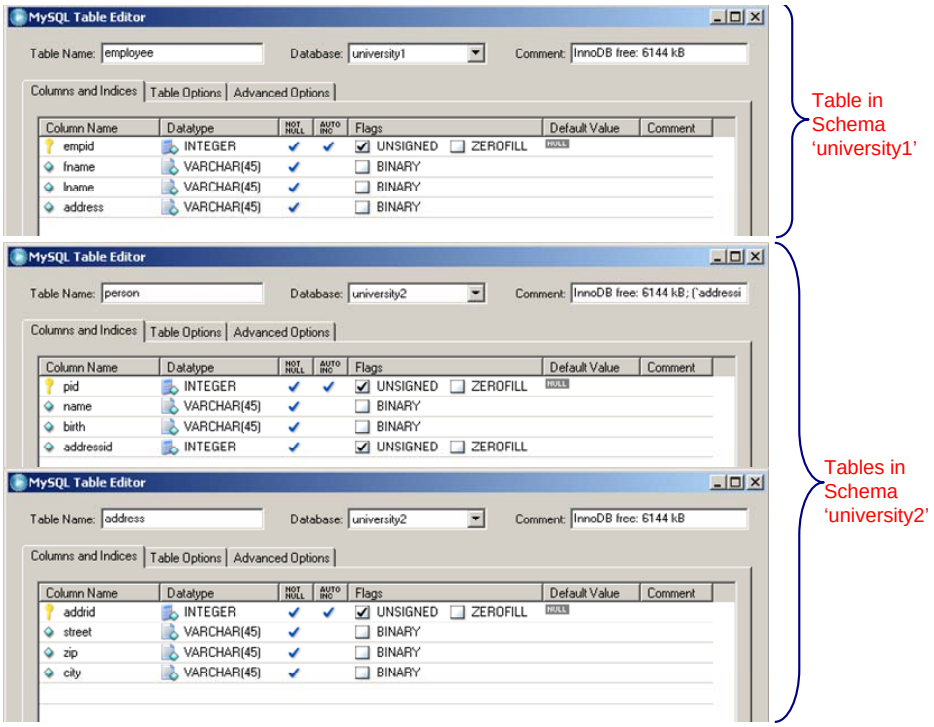


Fig. 5.9. Recipient and Donor Schemas in the Database

5.4.3 Matching Schemas

After displaying recipient and donor schemas, user selects the Match option from the menu. This option computes the similarities between the elements of these two schemas using the matching algorithms explained in Chapter 4. Results are filtered first based on the threshold value. As explained in Section 5.3.1 and shown in Figure 5.7, the value for threshold is set by the user before schema matching starts or default value is used. This value is used by schema matching for selecting the pairs, for which their similarities are above the threshold, and then the second filtering is applied using the selection strategy.

After applying these two filters, remaining similar pairs are displayed to the user for modification and validation. For the example university schemas, a screenshot of the system after schema matching is shown in Figure 5.11. In addition to a graph with "Similar To" edges

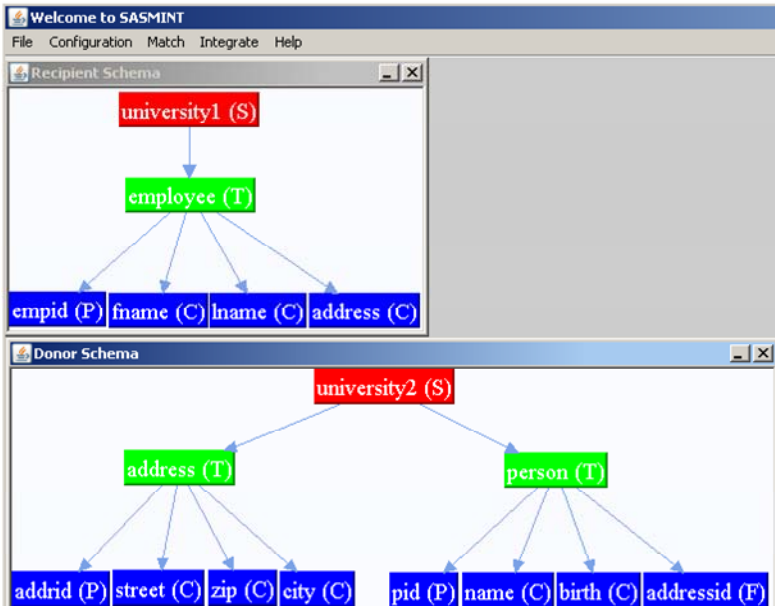


Fig. 5.10. Recipient and Donor Schemas Loaded

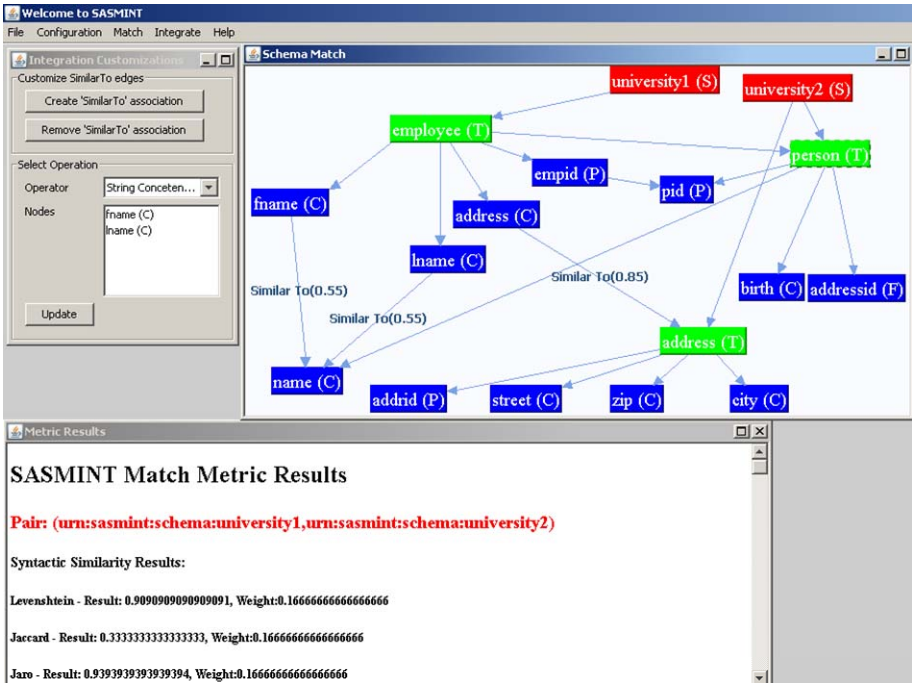


Fig. 5.11. Result of Schema Matching after User Validation

between matching nodes, detailed information about the results of each metric for pairs is also displayed on the GUI. User can modify the results on the graph. He can delete incorrect matches and introduce new ones and specify which kind of operation to use for combining *n* columns in 1-to-n or n-to-1 match cases. When the modifications are completed, the user can save the match results. A portion of the saved XML file for the example schemas is shown in Figure 5.12. This file can be used by a query processor to rewrite the query in terms of local schemas.

```
……..
……..
<graph:snode graph:id="urn:sasmint:column:university1:employee:empid" graph:name="empid“
        graph:schema="university1" graph:table="employee" graph:type="COLUMN“ graph:pkColumn=“yes“ />
<graph:snode graph:id="urn:sasmint:column:university1:employee:fname" graph:name="fname"
        graph:schema="university1" graph:table="employee" graph:type="COLUMN"/>
<graph:snode graph:id="urn:sasmint:column:university1:employee:lname" graph:name="lname"
        graph:schema="university1" graph:table="employee" graph:type="COLUMN"/>
<graph:snode graph:id="urn:sasmint:column:university1:employee:address" graph:name="address"
        graph:schema="university1" graph:table="employee" graph:type="COLUMN"/>
<graph:snode graph:id="urn:sasmint:schema:university2" graph:name="university2" graph:type="SCHEMA"/>
<graph:snode graph:id="urn:sasmint:table:university2:address" graph:name="address" graph:schema="university2"
         graph:type="TABLE"/>
……..
……..
<graph:sedge graph:id="urn:sasmint:hastable:683bb557-0d3d-4d2f-a3b2-840f3065254a“
         graph:sourceNodeId="urn:sasmint:schema:university1“
         graph:targetNodeId="urn:sasmint:table:university1:employee" graph:type="HASTABLE"/>
<graph:sedge graph:id="urn:sasmint:hascolumn:79b02ca1-9572-41af-b9d7-7db0b4d3390f“
         graph:sourceNodeId="urn:sasmint:table:university1:employee“
         graph:targetNodeId="urn:sasmint:column:university1:employee:empid" graph:type="HASCOLUMN"/>
……..
……..
<graph:sedge graph:id="urn:sasmint:similarTo:3a38c020-d184-4f11-8297-d58125f1784d"
         graph:sourceNodeId="urn:sasmint:column:university1:employee:lname"
         graph:targetNodeId="urn:sasmint:column:university2:person:name" graph:type="SIMILARTO">
                <graph:similarity>0.5142156907717128</graph:similarity>
</graph:sedge>
<graph:sedge graph:id="urn:sasmint:similarTo:e6738592-356d-4b97-9841-3e90b2cde8aa"
         graph:sourceNodeId="urn:sasmint:column:university1:employee:fname"
         graph:targetNodeId="urn:sasmint:column:university2:person:name" graph:type="SIMILARTO">
                <<graph:similarity>0.5142156907717128</graph:similarity>

</graph:sedge>
……..
……..
```

Fig. 5.12. Result of Schema Matching in XML format

5.4.4 Integrating Schemas

After validating the results of schema matching, the user can continue with the schema integration step. Schema integration applies a number of heuristic rules, explained in Chapter 4, in order to automatically generate an integrated schema of recipient and donor schemas. For the example university schemas, SASMINT automatically produces the integrated schema shown in Figure 5.13. Both the graph and XML (based on SDML) representations of the integrated schema are generated by SASMINT. The XML representation defines the derivations used to generate the elements of the integrated schema from the elements of input

schemas. User can modify the generated XML result and then save it. Since for integrating schemas of all nodes in a network, two schemas are integrated at a time, the XML file will expand after each integration process, with the definitions of the new nodes and edges as well as their derivations.

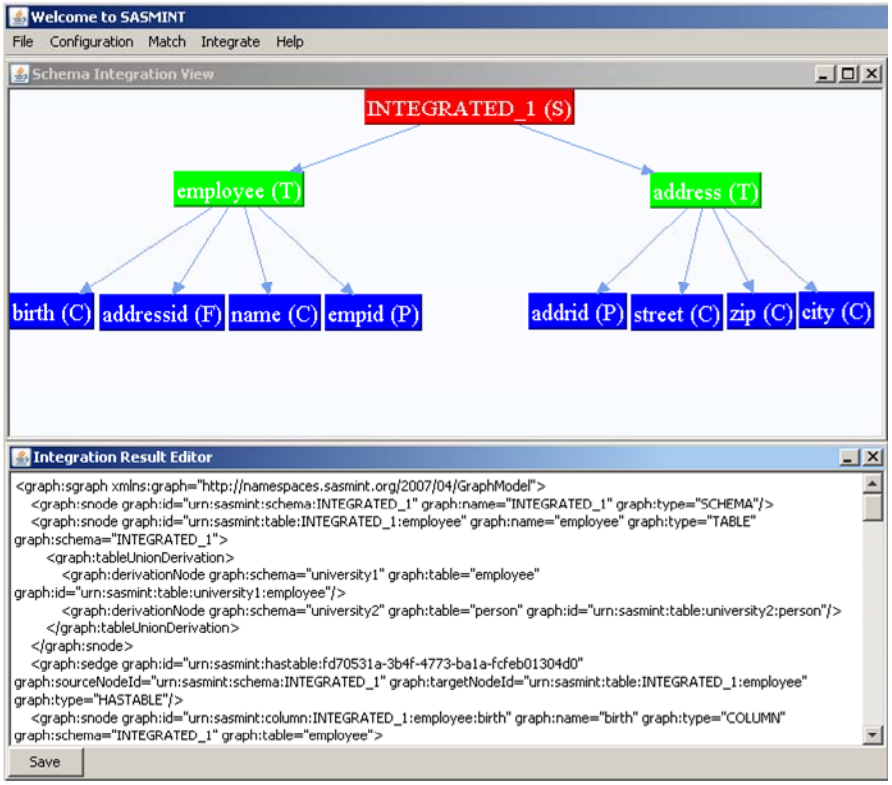


Fig. 5.13. Result of Schema Integration

5.5 Conclusions

In this chapter, we provide the details of the SASMINT implementation, which has been realized in order to both verify and to serve as a proof of concept for our proposed approach. The implementation functionally comprises the processing steps (i.e. called phases), and is made available to the user by means of a GUI, which we consider to be an important aspect of the overall proposed system solution. We observed that most existing schema matching and/or integration tools lacked sophisticated GUI's, which is why we decided to implement a GUI for SASMINT.

The majority of our implementation efforts have been dedicated to implementing 1) The construction of GUI, and 2) The schema integration rules (heuristics)

For implementing the SASMINT system, the platform-independent Java programming language and runtime has been utilized. A number of additional Java libraries, IDE's and add-ons have also been used to enhance our implementation.

Chapter 6

Empirical validation of SASMINT

In order to measure the quality and performance of our approach to schema matching and integration, we have performed a number of experiments. These experiments consider and make use of schemas that include different types of schema heterogeneities, as addressed in Chapter 3. This chapter describes these experiments and their results. In this respect, in Section 6.1 we first address a number of related experiments performed by other main research efforts. A number of specific quality measures used for assessing the results of our schema matching and schema integration components of SASMINT are described next in Section 6.2. The main characteristics of test schemas are addressed in Section 6.3. The setup and details related to the performed experimental evaluations are given in Section 6.4. Our evaluation results are addressed in Sections 6.5 to 6.7. Finally, Section 6.8 concludes the chapter with a summary of evaluation results.

This chapter contains some research results, which were previously published in the Journal of Knowledge and Information Systems (Unal & Afsarmanesh, 2010).

6.1 Schema Matching Evaluations in Related Research

The evaluation performed in most existing schema matching research does not use any benchmark; rather they each use their own test schemas in evaluating specific aspects of their proposed system.

A comparison of different evaluations introduced by different research for schema matching systems is provided in (Do et al., 2002). It specifies four different types of criteria to compare existing evaluations, including the evaluation of COMA (Do & Rahm, 2002), Cupid (Madhavan et al., 2001), Similarity Flooding (Melnik et al., 2002), SEMINT (Li & Clifton, 2000), and GLUE (Doan et al., 2002). These criteria include:

1) *Input:* Types of input data used, such as dictionaries used and schema specification.
2) *Output:* Information included in the match result, such as the mappings between different schema elements.
3) *Quality measures:* Measures used to assess the accuracy of the match result.
4) *Effort:* Types of needed manual effort measured in evaluations, such as pre-match and post-match efforts.

As (Do et al., 2002) concludes, it is difficult to compare results of different schema matching evaluations with each other, as these evaluations have been carried out in different ways and aimed at specific features. Authors further point at the requirement for a schema matching

benchmark to make the comparison of results of different research evaluations possible. Such a generic benchmark has however not yet been defined and/or considered in any research work. So far, only a benchmark for evaluating the systems, which match XML Schemas is proposed in (Duchateau et al., 2007). This benchmark, called XBenchMatch, consists of quality measures for both schema matching and schema integration. It also provides some evaluation of the matching performance. In XBenchMatch it is assumed that for evaluating the quality of schema matching, mappings must be given as XML path correspondences (e.g. person.person_name - person.lastName). Furthermore, for evaluating the quality of "integrated schema", a number of measures are introduced as a part of XBenchMatch. However, these measures assume that also the correct (ideal) integrated schema is provided to the XBenchMatch. As such, the integrated schema which is generated as the output of the schema integration tool is compared against the ideal integrated schema. Both of these schemas need to be in the XML Schema format. For the purpose of evaluating the quality of "schema matching", XBenchMatch applies the four measures of *Precision, Recall, F-measure, and Overall,* as most other evaluation approaches also apply some of these methods. Detailed description of these measures is provided in the next section.

In summary, most evaluation approaches consider only the quality of schema matching. Although the XBenchMatch prototype measures the quality of both the schema matching and schema integration, it can only support the XML Schema formats. Furthermore, there are some assumptions of XBenchMatch (e.g. the availability of the ideal integrated schema) as explained in the previous paragraph, which makes the general use of this benchmark difficult. Since SASMINT works with relational schemas and due to other reasons addressed above, we could not apply XBenchMatch for the evaluation of SASMINT. Nevertheless, as addressed in Section 6.2 below in details, nearly all measures introduced in other competitive research are considered and applied for validation of SASMINT.

6.2 Quality Measures Used for Evaluating SASMINT

The main goal of SASMINT is to automate the schema matching and integration processes to the extent possible. In other words, our main concern for SASMINT is its effectiveness, in how accurately the system can identify the matching pairs and generate the integrated schema automatically. For this reason, we consider only the quality and accuracy measures in our evaluations of the SASMINT system, and do not take into account the time performance related measures and assessment. Performance measures depend on the underlying environment and the technologies used, and thus it is challenging to obtain neutral objective evaluations. Furthermore, for schema matching and integration, when performance is considered, it is not only related to how fast the system works but also how much time the user spends correcting the results manually. Therefore, when the system produces more accurate results, the user needs to spend less manual time and the overall performance increases. Therefore, the accuracy aim of SASMINT also improves the performance of its schema matching and schema integration.

We apply two types of quality measures in our experiments: 1) quality measures for schema matching, and 2) quality measures for schema integration. Details of these measures are provided in the next sub-section.

6.2.1 Quality Measures for Schema Matching

Similar to most other schema matching evaluations, we used the concepts of precision and recall from the information retrieval field (Cleverdon & Keen, 1966) for measuring the quality of schema matching. Precision (P) and Recall (R) are computed as follows:

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

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

Although precision and recall measures are widely used for a variety of evaluation purposes, neither of them alone can accurately assess the match quality. For instance, recall can be increased by returning all pairs as similar, but increasing the number of false positives and thus decreasing the precision. Therefore, a measure combining precision and recall is better suited for accuracy evaluation. F-measure (Rijsbergen, 1979) is one such measure, combining recall and precision using the following formula. As such the higher the f-measure value is, the better is the system.

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

Another such measure, called Overall, is proposed by (Melnik et al., 2002). It is different from f-measure in that overall takes into account the amount of work needed to correct the results, namely to add the relevant needed matches that have not been discovered (false negatives) and to remove those matchers, which are incorrect but have been extracted by the matcher (false positives). Overall is always lower than f-measure, and if the precision is lower than 0.5, the result for overall becomes negative (Melnik et al., 2002) (Do et al., 2002). Overall, represented by O, and also called as accuracy, is defined by the following formula. As such the higher the overall value is, the better is the system.

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

As an example, assume that an automatic schema matching system correctly identifies 10 matches out of 25 real matches that can be identified manually by the user, and incorrectly identifies 4 other matches. In this case, the number of true positives (x) is 10, false negatives (y) is 25-10 = 15, and false positives (z) is 4. As a result, the system has the following precision, recall, f-measure, and overall values:

$$P = \frac{10}{10+4} = 0.71 \quad R = \frac{10}{10+15} = 0.40$$

$$F = \frac{2}{1/0.71 + 1/0.40} = 0.51 \qquad O = 0.40 * (2 - (1/0.71)) = 0.24$$

6.2.2 Quality Measures for Schema Integration

Quality measures used for the assessment of schema integration in SASMINT benefit from the ideas presented in (Batini et al., 1986). Schema merging and restructuring processes described in (Batini et al., 1986) aim at improving the resulting schema with respect to the following three qualities:

1) *Completeness:* Merged or integrated schema must cover concepts of all participating schemas.
2) *Minimality:* If the same concept is represented in more than one participating schemas, then the integrated schema must contain only a single representation of this concept. In other words, redundancies must be eliminated.
3) *Understandability:* Resulting integrated schema must be easily understandable by the user.

In evaluation of SASMINT, we are interested in quantitative objective measures. For this reason, we only consider measuring the *completeness* and *minimality* which will produce objective results. The *understandability* of SASMINT, while not measured rigorously, was satisfactory for the empirical tests we performed in the lab. The two measures of completeness and minimality applied to SASMINT are inspired by (Batini et al., 1986). However, within each of these measures, we have introduced two other measures for *key completeness* and *key minimality* to validate the generated primary and foreign keys when measuring the quality of SASMINT's schema integration approach. We belive that these added measures, which are missing from Batini's approach, are required for proper validation of schema integration. These measures are explained below:

- **Completeness Measure:** In the resulting integrated schema, all concepts (i.e. tables and columns in the relational schema) of both the donor and recipient schemas must be covered. Completeness measure determines how much this goal has been achieved.

  Therefore, $\forall\ c_i \in \{c_1, c_2, ..c_k\}$, where c_i is a concept in the donor or recipient schema and k is the total number of concepts in that donor or recipient schemas, $\exists c_j \in \{c_1, c_2, ..c_l\}$ where c_j is a concept of the integrated schema and $c_j \supseteq c_i$ and l is the number of concepts in the integrated schema. Taking this definition as the base, completeness of an integrated schema in SASMINT is measured using the following formula:

  $$m_{completeness} = \frac{n_{complete}}{n_{total}},$$

  where $n_{complete}$ is the number of concepts of recipient and donor schemas that are covered in the integrated schema and n_{total} (also l above) is the total number of concepts involved in donor and recipient schemas.

  Schema integration in SASMINT also handles primary and foreign keys, which will be referred to as "keys" from this point onward. Therefore, another completeness measure, called *key completeness*, is also defined for SASMINT to measure how many of the keys of the recipient and donor schemas are covered in the integrated

schema. Given that $n_{completeKey}$ is the number of keys of recipient and donor schemas that are covered in the integrated schema and $n_{totalKey}$ is the total number of keys involved in donor and recipient schemas, the following formula measures the key completeness, $m_{completenessKey}$, of an integrated schema in SASMINT:

$$m_{completenessKey} = \frac{n_{completeKey}}{n_{totalKey}}$$

- **Minimality Measure:** The amount of redundancy in the resulting integrated schema must be minimal to the extent possible. Each joint and/or related concept of the donor and recipient schemas shall appear only once in the integrated schema. Namely, if the donor and recipient schemas have common concepts, only one of them must be represented in the integrated schema. Minimality measure identifies how many redundant concepts exist in the integrated schema.
Suppose that $\exists c_i \in \{c_1, c_2, ..c_k\}$, where c_i is a concept of the donor schema and k its total number of concepts, and $\exists c_j \in \{c_1, c_2, ..c_l\}$, where c_j is a concept of the recipient schema and l its total number of concepts. If $\exists c_x, c_y \in \{c_1, c_2, ..c_m\}$, where c_x and c_y are concepts of the integrated schema and m its total number of concepts, such that $c_i \equiv c_j \equiv c_x \equiv c_y$, then either c_x or c_y is redundant. Following formula is used to calculate the amount of redundancy in an integrated schema:

$$m_{redundancy} = \frac{n_{redundant}}{n_{total}},$$

where $n_{redundant}$ is the number of redundant concepts in the integrated schema and n_{total} is the total number of concepts introduced in the donor and recipient schemas.
Based on this formula, we derive the following formula to measure the minimality of the SASMINT integrated schema.

$$m_{minimality} = 1 - \frac{n_{redundant}}{n_{total}}$$

Similar to the case of completeness measure, another minimality measure, called *key minimality*, is also defined for SASMINT to determine if the resulting integrated schema is minimal considering its primary and foreign keys. Key minimality, $m_{minimalityKey}$, is measured using the following formula:

$$m_{minimalityKey} = 1 - \frac{n_{redundantKey}}{n_{totalKey}}$$

where the $n_{redundantKey}$ is the number of redundant primary and foreign keys in the integrated schema and the $n_{totalKey}$ is the total number of such keys introduced in the donor and recipient schemas.

6.3 Test Schemas

We have carried out the experimental evaluation of SASMINT using six pairs (donor and recipient) of "test schemas", characteristics of each of which are shown in Table 6.1 and the six pairs of schemas are represented in Appendix D. As for the evaluation of **schema matching**, each pair was matched by the SASMINT, and then the results were compared against the correct matches shown in Table 6.2. We carried out the same tests for schema matching in COMA++ (a leading competitor) and compared its results with the results of SASMINT. On the other hand, for evaluation of **schema integration**, three pairs of schemas all from the university domain (in Table 6.1) were integrated. Moreover, in order to evaluate the Sampler component, first five schema pairs were used in the Sampler tests. Details of these tests are provided in the next sections.

Table 6.1. Characteristics of Test Schemas

Test Schema Pair #	Short Name	Domain	Donor/ Recipient	Number of Tables	Number of columns
1	PO	Purchase Order	Recipient	5	27
			Donor	5	25
2	Hotel	Hotel	Recipient	6	21
			Donor	5	14
3	SDB	Biology	Recipient	9	21
			Donor	9	22
4	Univ1	University	Recipient	9	30
			Donor	5	22
5	Univ2	University	Recipient	9	38
			Donor	7	27
6	Univ3	University	Recipient	5	17
			Donor	3	10

We used schemas from four different domains: Schema Pair#1 contains two purchase order schemas that we generated ourselves. Schema Pair#2 consists of two hotel schemas. We modified the hotel schemas used for MAPONTO (An et al., 2006) evaluation tests. Similarly, in Schema Pair#3, we used a modified version of MAPONTO SDB schemas from the biology domain. In Schema Pair#4, we used MAPONTO schemas from the university domain, again after modifying them. Schema Pair#5 consists of university schemas that we generated. As Schema Pair#6, we modified the test schemas of Similarity Flooding (Melnik et al., 2002) from the university domain. We intentionally selected three pairs from the university domain in order to also use them for the schema integration evaluation. Therefore, the schema integration tests integrated six schemas from the university domain.

The correct matches represented in Table 6.2 are matches that are generated manually by ourselves. These constitute the source for verification of correctness of automatic matchings.

Table 6.2. Correct Matches between Schema Pairs

Schema Pair#	Type	Correct Matches
1	**table-table**	purchase_order=po, customer=buyer, product=item
	Colum-column	purchase_order:custNo=po:buyer_no, purchase_order:deliverDate=po:deliver_date, purchase_order:deliverCity=po:deliver_city, purchase_order:deliverStreet=po:deliver_street, purchase_order:deliverZip=po:deliver_zip, purchase_order:purchaseOrderNo=po:po_no, customer:street=buyer:buyer_street, customer:name=buyer:f_name, customer:telephone=buyer:phone, customer:city=buyer:buyer_city, customer:custNo=buyer:buyer_No, customer:name=buyer:l_name, customer:zip=buyer:buyer_zip, product:stock=item:stock, product:price=item:cost, product:productNo=item:item:item_no, product:productName=item:item_name
2	**table-table**	one_room=room, suite=room, town_house=room, num_beds=num_beds_attribute, on_floor=on_floor_attribute, smoking_preference=smoking_attribute
	column-column	one_room:roomNum=room:roomNum, one_room:hasNumBedsAttribID=room:numBedsAttribID, one_room:hasOnFloorAttribID=room:onFloorAttribID, one_room:hasSmokingPreferenceAttribID=room:smokingOrNoAttribID, one_room:oneRoomID=room:roomID, suite:suiteID=room:roomID, suite:roomNum= room:roomNum, suite:hasNumBedsAttribID=room:numBedsAttribID, suite:hasOnFloorAttribID=room.onFloorAttribID, suite:hasSmokingPreferenceAttribID=room.smokingOrNoAttribID, town_house:townHouseID=room.roomID, townhouse:roomNum=room.roomNum, town_house:hasNumBedsAttribID=room:numBedsAttribID, town_house:hasOnFloorAttribID=room:onFloorAttribID, town_house:hasSmokingPreferenceAttribID=room:smokingOrNoAttribID, num_beds:numBedsID=num_beds_attribute:numBedsAttributeID, num_beds:numBedsAttrib=num_beds_attribute:numBedsAttrib, on_floor:onFloorID=on_floor_attribute:onFloorAttributeID, on_floor:onFloorAttrib=on_floor_attribute:onFloorAttrib, smoking_preference:smokingPreferenceID=smoking_attribute:smokingAttributeID, smoking_preference:smokingPreferenceAttrib=smoking_attribute:smokingAttrib
3	**table-table**	diagnoses=diagnoses, donor=donor, sample=sample, family_history=family_history, life_style_factors=life_style_factors, lab_test=lab_test, medications=medications, animal_donor=donor, human_donor=donor
	column-column	animal_donor:strain=donor:strain, diagnoses:diagID=diagnoses:diagID, donor:gender=donor:gender, donor:id=donor:id, donor:species=donor:species, family_history:histID=family_history:histID, human_donor:dob=donor:dob, lab_test:testID=lab_test:testID, life_style_factors:factID=life_style_factors:factID, medications:medicID=medications:medicID, sample:name=sample:name, animal_donor:species=donor:species, animal_donor:animalID=donor:id, human_donor:gender=donor:gender, human_donor:humanID=donor:id
4	**table-table**	course=course, student=student, faculty_member=academic_staff

Schema Pair#	Type	Correct Matches
	column-table	faculty_member:researchInterest=areasOfInterest
	Column-column	faculty_member:email=academic_staff:email, faculty_member:faculty_member_id=academic_staff:academic_staff_id, faculty_member:personName=academic_staff:name, course:number=course:courseNumber, course:courseTitle=course:courseTitle, course:instructor=course:instructor, course:prerequisites=course:prerequisite, course:description=course:description, student:studentName=student:student_name, student:email=student:email, student:advisor=student:supervisor, student:student_id=student:student_id,
5	**table-table**	university=acedemic_institution, program=program, academic_programme=academic_programme, department=department, course=academic_course, academic_staff_member=university_academic_instructor
	column-column	university:university_ID= academic_institution:academic_institution_ID, university:UNIVERSITY_NAME= academic_institution:ACADEMIC_INSTITUTION_NAME, university:UNIVERSITY_WEBSITE=academic_institution:ACADEMIC_INSTITUTION_WEBSITE, program:program_ID= program:program_ID, program:PROGRAM_NAME=program:PROGRAM_NAME, program:PROGRAM_DESC=program:PROGRAM_DESC, academic_programme:academic_programme_ID=academic_programme: academic_programme_ID, academic_programme:ACADEMIC_YEAR=academic_programme:YEAR, academic_programme:ACADEMIC_SEMESTER=academic_programme:SEMESTER, academic_programme:PROGRAM_REF= academic_programme:PROGRAM_REF, department:department_ID=department:department_ID, department:DEPT_NAME=department:DEPT_NAME, course:course_ID=academic_course:academic_course_ID, course:COURSE_NAME=academic_course:ACADEMIC_COURSE_NAME, course:COURSE_CREDITS = academic_course:ACADEMIC_COURSE_CREDITS, course:COURSE_PROVIDER=academic_course:ACADEMIC_COURSE_PROVIDER, course:COURSE_INSTRUCTOR= academic_course:ACADEMIC_COURSE_INSTRUCTOR, academic_staff_member:academic_staff_member_ID = university_academic_instructor:university_academic_instructor_ID, academic_staff_member:STAFF_NAME=university_academic_instructor:NAME , academic_staff_member:STAFF_EMAIL = university_academic_instructor:ELECTRONIC_MAIL, academic_staff_member:STAFF_PHONE=university_academic_instructor.TELEPHONE
6	**table-table**	professor=professor, student=student, workson=workson
	table-column	address=professor:address
	column-column	professor:Id=professor:Id, professor:Name=professor:Name, professor:Sal=professor:Salary, student:Name=student:Name, student:GPA=student:GradePointAverage, student:Yr=student:Year, workson:Name=workson:StudentName, workson:Proj=workson:Project

6.4 Setup for the Experimental Evaluation

We compared the "schema matching" component of SASMINT against one of the state of the art system, COMA++ (Aumueller et al., 2005). We selected COMA++ research prototype, because it is the most complete schema matching tool so far developed, consisting of a library of variety of matching algorithms and a sophisticated GUI. SASMINT and COMA++ are comparable, since they both support matching of relational schemas and aim at providing similar functionalities. Of course not all algorithms or metrics that these two systems apply are the same. Furthermore, how they combine the results of different algorithms is not the same either. Output of the schema matching is given in the range [0-1] in both systems. However, it is not clear in COMA++, in what format the results of schema matching are stored internally. In other words, COMA+++ has an internal repository where the results are stored, but how the results are represented there is not clear.

Before starting the evaluation tasks, we inserted a number of abbreviations and their long forms into the abbreviation lists of both systems. One important difference between SASMINT and COMA++ is that SASMINT uses WordNet for semantic matching, whereas COMA++ requires the user to add all needed synonyms in the schema domains manually. Since WordNet might not contain all semantic relationships among the concepts of schemas, in order to make a fair comparison, we did not make any addition to COMA++'s default synonyms list, to make a fair comparison. Furthermore, COMA++ uses only the synonymy relationship; on the other hand, SASMINT also makes use of the IS-A relationships as well as gloss overlaps, which are available in the WordNet dictionary.

Representation of schemas through the GUI is also different for the two systems. COMA++ does not explicitly show foreign keys. Instead of showing the foreign key column, it displays the table that is pointed by the foreign key. However, in some cases this functionality of COMA++ does not work as expected.

Several different metrics or algorithms are considered and combined in both systems, in the manner that is explained below:

- **For SASMINT:** We selected the default strategy of SASMINT for combining the algorithms, which is the weighted sum of them with equal weights applied to each algorithm in each group of syntactic, semantic, and structure matching. Although not the default approach, rather assigning appropriate weights for each match task would give better results, we decided to use SASMINT's default strategy in order to make a fair comparison with COMA++. In other words, in real practice, the results of SASMINT would be better than what they are in these tests. Sampler could help the user to identify appropriate weights for the linguistic matching algorithms. The reported evaluation results in Section 6.5 and Appendix E are without applying the Sampler component of SASMINT. Results of experiments showing how Sampler can accomplish this improvement of results accuracy, i.e. how these weights affect the match results, are addressed in Section 6.7.

- **For COMA++:** We used the default matching strategy of COMA++, which is called COMA. The COMA matcher combines the *name*, *path*, *leaves*, *parents*, and *siblings* matchers, by averaging them. In their tests, this combination was the winner and that is why we selected it.

We used the default threshold, which is 0.5, in the experiments. As for the selection of match results, we used two different approaches that we call as "select all above threshold" and "select max above threshold", as detailed below. Please note that while the results of

"select all above threshold" are presented in Section 6.5, the results of "select max above threshold" are presented in Appendix E.

1) **Select All above Threshold:** Selecting all matched pairs that have the similarity above a certain threshold value.

2) **Select Max above Threshold:** Selecting the pairs with the maximum similarity. In other words, whenever there is more than one concept matching a single concept in a schema, the one with the highest similarity is selected as the matching candidate. SASMINT and COMA++ use different strategies for selecting the maximal similar pairs. SASMINT's approach is explained in Chapter 4. COMA++'s default strategy works as follows: When there is more than one match to the same concept, the one with the highest similarity is selected if the difference between the similarity values is more than 0.0080.

We also carried out tests in order to validate the Sampler component that helps to identify appropriate weights for each linguistic matching algorithm in the schema matching process. The results of these tests are presented in Section 6.6. The first five schema pairs (Schema Pairs #1, 2, 3, 4, and 5) were selected as the test schemas when evaluating Sampler.

Although we compared SASMINT with COMA++ for the purpose of schema matching, we could not carry out functionality comparison for schema integration between them. COMA++ provides a simple schema merging functionality, but it is limited and not comparable to SASMINT's schema integration. To the best of our knowledge, there is no other system supporting both schema matching and schema integration. Therefore, we evaluated the integration component of SASMINT alone. For this purpose, we used the six schemas from the university domain, introduced as Schema Pair#4, 5, and 6. Since the aim of schema integration is integrating two schemas at a time, based on the correspondences between them, we corrected the wrong or missing matches after the schema matching step and then continued with the integration process.

6.5 Evaluation of Schema Matching – For "select all above threshold" strategy

In the first experiment that we performed to evaluate SASMINT and compare it with COMA++, we used the "select all above threshold" strategy. We present the results of this experiment in Figures 6.1 through 6.8. Correspondingly, we provide detailed explanations about the four comparison results of precision, recall, f-measure, and overall in the following paragraphs 6.5.1 to 6.5.4. Although the results gained from applying this strategy are worse than the "select max above threshold" strategy, this strategy is important when there is a need for suggesting multiple candidates for each schema element and leaving it to the user to identify the correct match among the alternatives. Namely, instead of proposing only one matched candidate for each schema element, which could be incorrect, the system suggests all possible match candidates, which makes it easier for the user to determine the final match result.

6.5.1 Evaluation of Schema Matching Using Precision

Precision shows how correct the system works. Precision values for COMA++ and SASMINT are shown in Figure 6.1 and Figure 6.2 respectively. Since in the "select all above threshold" strategy, all match pairs with similarity above the threshold are selected, the number of false positives was high for some schema pairs. Especially for schemas that consisted element names with more than one token, precision was low. In our test cases, these schemas are the purchase order schemas (Schema Pair#1), university schemas of Maponto (Schema Pair#4), and the university schema that we generated ourselves (Schema Pair#5). In these cases, the low precision was due to the fact that for element names containing similar tokens, although the whole names were different, the final similarity result was usually above the threshold. Furthermore, the systems interpreted and treated all tokens equally, while some tokens had none or little effect in the meaning. For example, "deliverDate" and "deliver_zip" were identified as similar because both names contained the token "deliver". However, the first one is the name of the column that contains the date of delivery, whereas, the second one is the name of the column that contains the zip code information. In such situations, SASMINT and COMA++ both found similarity values around 0.5. These cases could have been prevented by raising the threshold value, but then some correct matches could have been also missed. When precision was considered, SASMINT achieved almost 9 times better than COMA++ for the Hotel schemas test case. For other schemas, except for Schema Pair#6 from the university domain, for which COMA++ achieved just a little bit better (around 1.05 times), SASMINT achieved on average 2 times better than COMA++. Precision of SASMINT was on the average 0.58, whereas that of COMA++ was 0.26. This result was because of the high number of false positives identified by COMA++. In other words, COMA++ identified high number of irrelevant matches, which can be a bigger problem when schemas being compared are large.

6.5.2 Evaluation of Schema Matching Using Recall

Recall shows how well the system finds all true matches and thus it indicates the completeness of the applied system. The average recall for COMA++ was 0.92, whereas for SASMINT it was 0.85. Figures 6.3 and 6.4 show the recall values for COMA++ and SASMINT

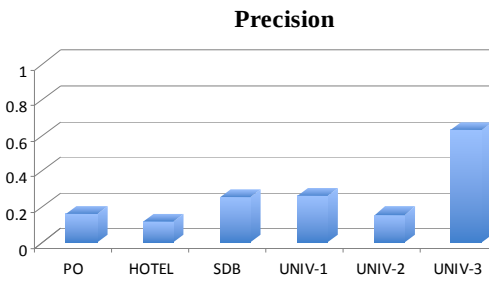


Fig. 6.1. Precision values for **COMA++** - select all above threshold strategy

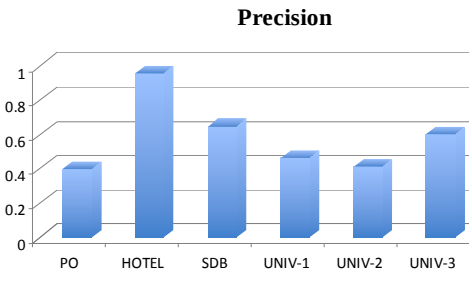


Fig. 6.2. Precision values for **SASMINT** - select all above threshold strategy

respectively. For UNIV-3 schemas, they both had the recall value of 1.0 and for SDB schemas, SASMINT was 1.14 times better than COMA++. For the remaining schemas, which were purchase order, hotel, UNIV-1, and UNIV-2, COMA++ achieved a bit (on the average 1.17 times) higher than SASMINT. However, it should be noted that this happened at the expense of very low precision values for COMA++. That means, in order to achieve just a bit higher recall values, COMA++ sacrificed the precision, resulting in very low precision values for these test cases, as indicated in Figures 6.1 and 6.2. This is due to the fact that there is an inverse relationship between precision and recall. Since COMA++ tries to find all possible matches, it also identifies a large number of false positive matches, which decrease the precision. SASMINT missed some of the correct matches, mostly due to low semantic similarity values that it could compute for some name pairs, such as (product, item) and (suite, room). Especially the gloss-based measure was not as successful as expected. Since the last version of WordNet (3.0) is not available yet for the Windows operating system, we had to use the previous version (2.0) of WordNet. We think that when the new version is ready, WordNet will provide more types of semantic relationships, and therefore the semantic similarity values for both path-based and gloss-based measures of SASMINT will be much more enhanced.

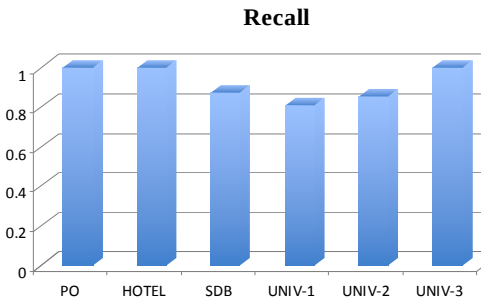


Fig. 6.3. Recall values for **COMA++** - select all above threshold strategy

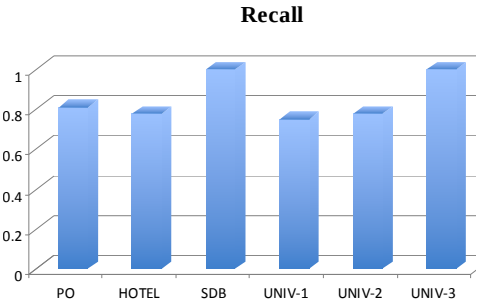


Fig. 6.4. Recall values for **SASMINT** - select all above threshold strategy

6.5.3 Evaluation of Schema Matching Using F-Measure

As stated before, f-measure is used to combine the results of precision and recall. In other words, the higher the f-measure value, the better is the quality of the system. Most evaluation experiments in fact use f-measure as the measure to compare the systems, and not the individual precision and recall values. When f-measure is considered, the difference between SASMINT and COMA++ becomes clearer. This is due to the fact that f-measure considers both the precision and recall, and although recall values for COMA++ were a bit higher than those for SASMINT, precision of SASMINT was much better than that of COMA++, which results in higher f-measure values for SASMINT. As it is clear from the Figures 6.5 and 6.6, f-measure values for SASMINT were on average 2.2 times higher than those for COMA++ for all schema pairs, except the last schema pair (UNIV-3), for which they almost achieved the same. What can be inferred from these results is that the quality of results achieved by the SASMINT system is much higher than COMA++, considering the f-measure evaluation.

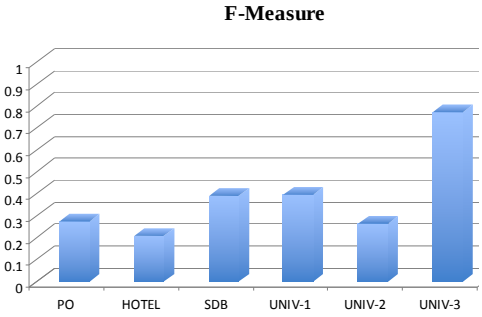


Fig. 6.5. F-measure values for **COMA++** - select all above threshold strategy

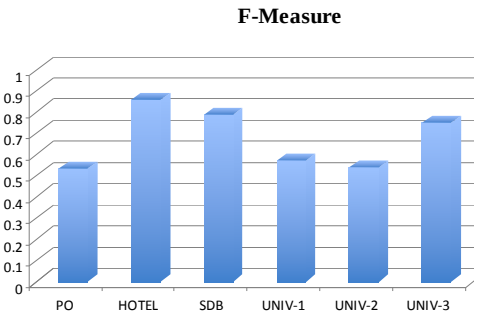


Fig. 6.6. F-measure values for **SASMINT** - select all above threshold strategy

6.5.4 Evaluation of Schema Matching Using Overall

Similar to f-measure, overall also represents a combination of precision and recall. Its value is smaller than both Precision and Recall and it can even have negative values, if the number of the false positives is more than the number of true positives. The overall indicator measures the overall accuracy of the system. It aims to identify how much manual effort is required in order to identify all correct matches. Overall values for SASMINT were consistently much higher than those for COMA++. In some cases, for example the hotel schemas, SASMINT achieved overall value around 0.7. In the case of UNIV-2 and purchase order schemas, on the other hand, it did not do very well because of the high number of false positive matches that SASMINT identified for these schemas. Since the number of false positive matches for COMA++ was very high, it had very low overall values, which means a lot of manual intervention by user is required in order to remove these wrongly identified matches. This result is very clear especially for the first five schema pairs (purchase order, hotel, SDB, and the UNIV-1, and UNIV-2 schemas). Evaluation results for COMA++ and SASMINT, based on the overall values are shown in Figures 6.7 and 6.8 respectively. Since the aim of such systems is to achieve the schema matching as automatically as possible, the amount of required human intervention is an important measure for comparing these systems. The lesser manual effort is required, the better the system is.

6.6 Evaluation of Schema Matching with Sampler

In order to evaluate the Sampler component, we carried out tests using the first five schema pairs introduced in Table 6.1. As explained before, Sampler is used to compute the weights only for linguistic matching algorithms. In test cases where the element names from two schemas were highly similar, we set the threshold to a value higher than 0.5. In other cases, we used the default threshold value, which was 0.5.

After setting the threshold value, we performed the tests using both equal weights for the linguistic matching algorithms and the weights suggested by Sampler for these algorithms. We

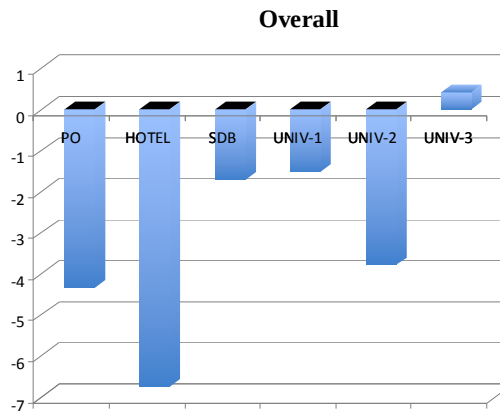


Fig. 6.7. Overall values for **COMA++** - select all above threshold strategy

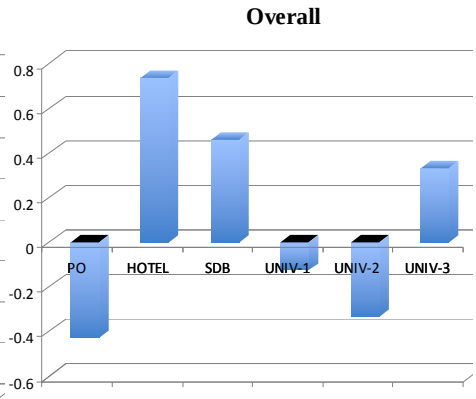


Fig. 6.8. Overall values for **SASMINT** - select all above threshold strategy

used the "select max above threshold" strategy for the Sampler tests. Furthermore, we did not use the last schema pair (schema pair#6) in the tests, because for this pair, precision, recall, f-measure, and overall values were already identified as 1 in the tests using the "select max above threshold" strategy, when equal weights were used. Details of tests with the Sampler component are explained below.

6.6.1 Test with Purchase Order Schemas-PO (Schema Pair#1)

In this test, we used the default threshold value, which was 0.5. We provided the similar pairs shown in Table 6.3 to Sampler, which computed the weights for semantic similarity algorithms shown in the same table. Results for precision, recall, f-measure, and overall were already high before the Sampler component was used. With the use of Sampler, (product, item) pair was correctly identified as similar, which was false negative before. As the result, Sampler helped to increase the values of recall, f-measure, and overall, as shown in Figure 6.9. Precision was 1 both before and after the use of the Sampler component.

Table 6.3. Similar Pairs and Computed weights for Schema Pair#1

Similar Pairs
Semantically Similar Pairs: customer - buyer product – item
Computed Weights
Wu and Palmer: 1.0 Gloss: 0.0

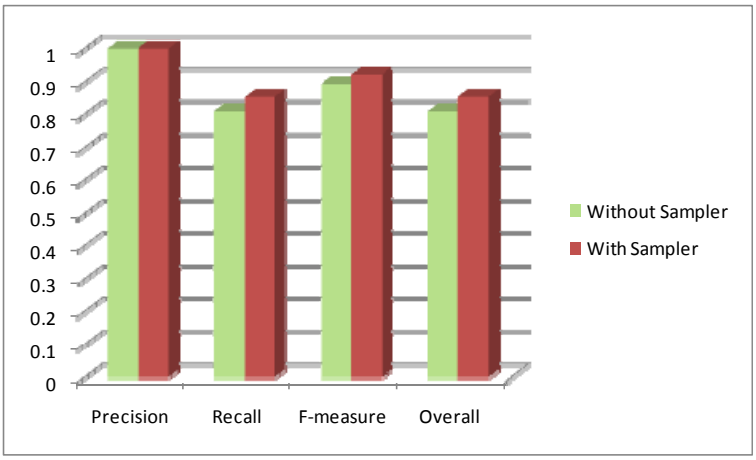


Fig. 6.9. Results of the Test with Schema Pair#1

6.6.2 Test with Hotel Schemas-Hotel (Schema Pair#2)

In this test, we set the threshold value as 0.7. We provided the similar pairs shown in Table 6.4 to Sampler, which computed the weights for syntactic similarity algorithms shown again in Table 6.4. When SASMINT used these weights for matching the hotel schemas, results for recall, f-measure, and overall were on the average 1.75 times (57%) better than the case without the use of Sampler. This result can be seen in Figure 6.10.

Table 6.4. Similar Pairs and Computed weights for Schema Pair#2

Similar Pairs
Syntactically Similar Pairs: smoking_Preference_Attrib - smoking_Attrib smoking_Preference_ID - smoking_Attribute_ID hasSmokingPreferenceAttribID - smokingOrNoAttribID on_floor - on_Floor_attribute numBedsID – numBedsAttributeID
Computed Weights
Levenshtein: 0.0 Jaccard: 0.11 LCS: 0.20 Monge-Elkan: 0.22 Jaro: 0.22 TF*IDF: 0.25

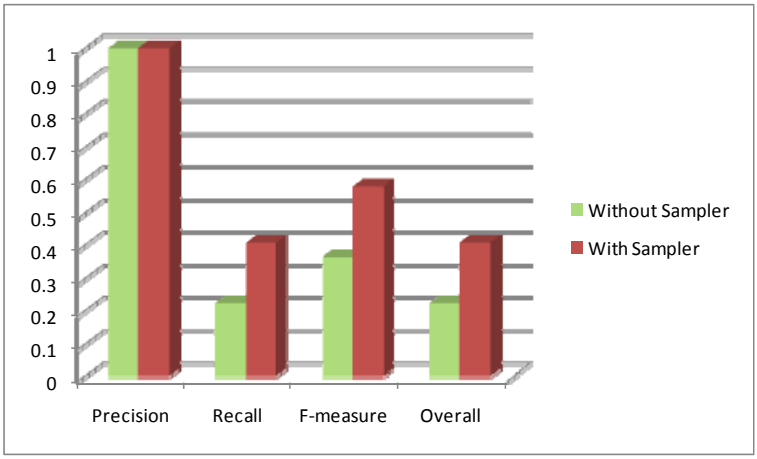


Fig. 6.10. Results of the Test with Schema Pair#2

6.6.3 Test with Biology Schemas-SDB (Schema Pair#3)

Two schemas (donor and recipient) in Schema Pair#3 use the same names for most of their schema elements. We set the threshold value to 0.9 and provided the two similar pairs of (animal_donor-donor) and (human_donor-donor). Sampler computed 1.0 for the weight of Monge-Elkan distance metric and 0.0 for other syntactic similarity metrics, as shown in Table 6.5. When we ran SASMINT with these weights, the results were as shown in Figure 6.11. There was a slight decrease in Precision when Sampler was used. This was due the two false positives (donor, donor_visit) and (donorID, donorVisitID). However, recall, f-measure, and overall were all improved.

Table 6.5. Similar Pairs and Computed weights for Schema Pair#3

Similar Pairs
Syntactically Similar Pairs: animal_donor - donor human_donor – donor
Computed Weights
Levenshtein: 0.0 Jaccard: 0.0 LCS: 0.0 Jaro: 0.0 TF*IDF: 0.0 Monge-Elkan: 1.0

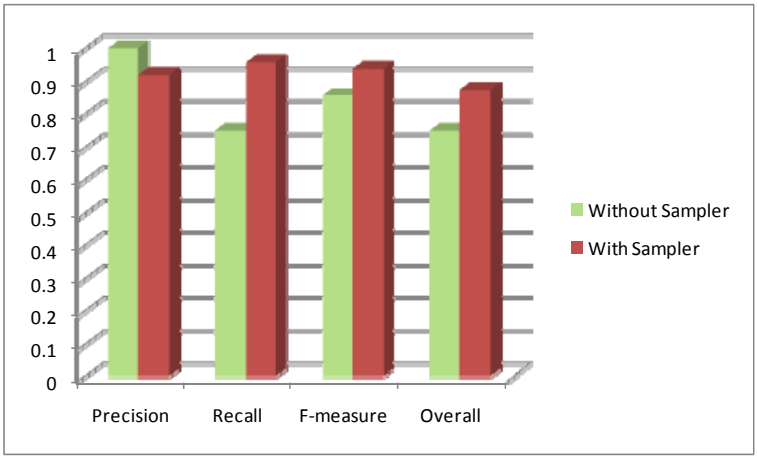


Fig. 6.11. Results of the Test with Schema Pair#3

6.6.4 Test with University Schemas-UNIV1 (Schema Pair#4)

For the test with these schema pairs, we set the threshold value as 0.7. We provided syntactically similar pairs, shown in Table 6.6. As shown in Figure 6.12, precision was slightly better before, whereas recall, f-measure, and overall values were higher with the use of Sampler. Since we provided Sampler (personName, name) as the syntactically similar pair, the *personName* column of the *faculty_member* table was successfully matched to the *name* column of the *academic_staff* table. However, at the same time, it incorrectly matched the *personName* column of the *faculty_member* and the *name* column of the *admin_staff* table. This in turn, increased the number of false positives, and thus slightly decreased the precision. However, since Sampler helped to identify more number of similar pairs, recall was much better than the case without the Sampler. As the result, f-measure and overall were better with the use of Sampler, as shown in Figure 6.12.

Table 6.6. Similar Pairs and Computed weights for Schema Pair#4

Similar Pairs
Syntactically Similar Pairs: number - courseNumber personName - name researchInterest – areasOfInterest
Computed Weights
Levenshtein: 0.0 Jaccard: 0.0 Jaro: 0.15 LCS: 0.15 TF*IDF: 0.3 Monge-Elkan: 0.4

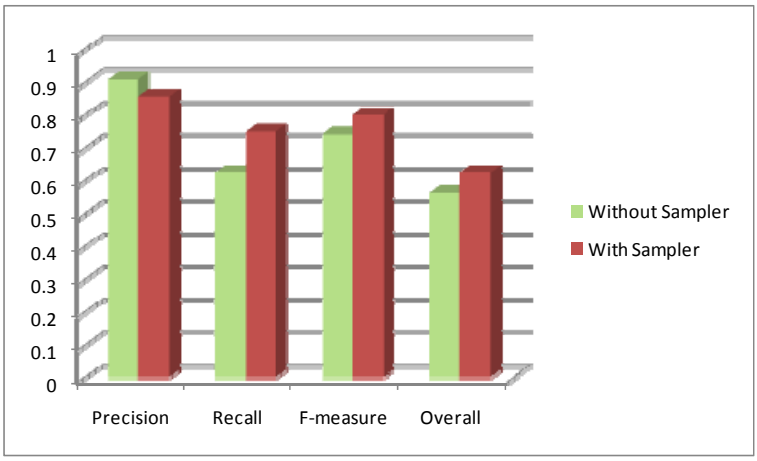


Fig. 6.12. Results of the Test with Schema Pair#4

6.6.5 Test with University Schemas-UNIV2 (Schema Pair#5)

In this test, we set the threshold value to 0.7 and provided the pairs shown in Table 6.7 to Sampler. Weights computed by Sampler for syntactic similarity algorithms are presented in Table 6.7. Similar to the case addressed in Section 6.7.4, with the use of Sampler the precision decreased because some new false positive pairs were introduced. For example, the *university_name* column of the *university* table and the *name* column of the *university_student* table were identified as similar, which was incorrect. However, since the value of recall was much higher when Sampler was used, f-measure and overall increased, as presented in Figure 6.13 also.

Table 6.7. Similar Pairs and Computed weights for Schema Pair#5

Similar Pairs
Syntactically Similar Pairs: academic_semester - semester course_id - academic_course_id course_instructor - academic_course_instructor staff_name - name course - academic_course
Computed Weights
Levenshtein: 0.0 Jaccard: 0.0 Jaro: 0.0 LCS: 0.18 Monge-Elkan: 0.35 TF*IDF: 0.47

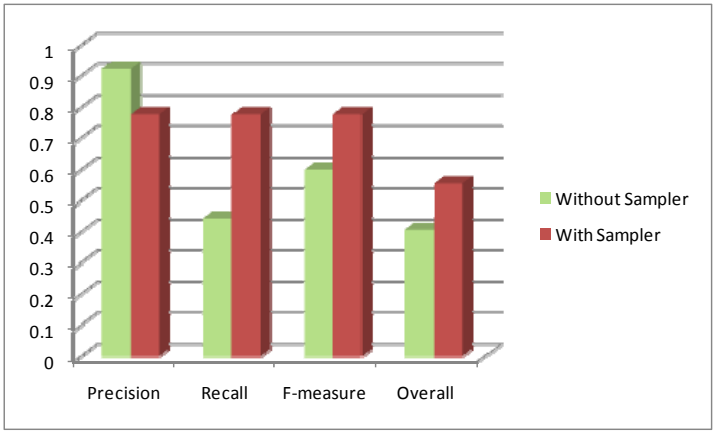


Fig. 6.13. Results of the Test with Schema Pair#5

6.7 Evaluation of Schema Integration Performance

In order to evaluate the schema integration component of SASMINT, we used schema pairs from the university domain. The three university schema pairs introduced in Table 6.1 which are Schema Pairs#4, 5, and 6 are used for this purpose. As addressed further below, please note that the Appendix F provides details of the steps of evaluation.

Figures 6.14 through 6.16 show the elements of these pairs. SASMINT integrates two schemas at a time, therefore, incrementally generating the final integrated schema. The steps we followed for integrating these six schemas are explained below. We have selected to start with larger schemas first, namely Schema Pair#5.

First Schema

academic_programme {academic_programme_ID, ACADEMIC_YEAR, ACADEMIC_SEMESTER,PROGRAM_REF}
academic_staff_member {academic_staff_member_ID, STAFF_NAME, STAFF_EMAIL, STAFF_PHONE, STAFF_FAX, STAFF_IDENTIFICATION_NUM, STAFF_BIRTHDATE}
campus {campus_ID, CAMPUS_NAME, CAMPUS_LOCATION, UNVCAMPUS}
course {course_ID, COURSE_NAME, COURSE_CREDITS, COURSE_PROVIDER, COURSE_INSTRUCTOR}
department {department_ID, DEPT_NAME, FACULTY_REF}
faculty {faculty_ID, FACULTY_NAME, DEAN_REF, UNIVERSITY_REF}
program {program_ID, PROGRAM_NAME, PROGRAM_DESC}
registration {registration_ID, REGISTRATION_ACADEMICSTAFFMEMBER_REF, REGISTRATION_COURSE_REF, REGISTRATION_ACADEMICPROGRAMME_REF}
university {university_ID, UNIVERSITY_NAME, UNIVERSITY_WEBSITE, UNIVERSITY_ESTABLISHMENT_DATE }

Second Schema

academic_course {academic_course_ID, ACADEMIC_COURSE_NAME, ACADEMIC_COURSE_CREDITS, ACADEMIC_COURSE_PROVIDER, ACADEMIC_COURSE_INSTRUCTOR}
academic_institution {academic_institution_ID, ACADEMIC_INSTITUTION_NAME, ACADEMIC_INSTITUTION_WEBSITE}
academic_programme {academic_programme_ID, YEAR, SEMESTER, PROGRAM_REF }
department {department_ID, DEPT_NAME, UNIVERSITY_REF}
program {program_ID, PROGRAM_NAME, PROGRAM_DESC}
university_academic_instructor {university_academic_instructor_ID, NAME, ELECTRONIC_MAIL, OFFICE_ADDRESS, TELEPHONE}
university_student {university_student_ID, NAME, ELECTRONIC_MAIL, TELEPHONE}

Fig. 6.14. Schema Pair#5 (UNIV-2)

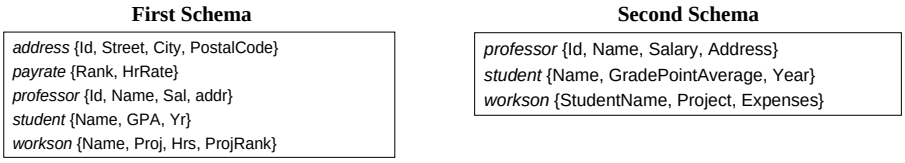


Fig. 6.15. Schema Pair#6 (UNIV-3)

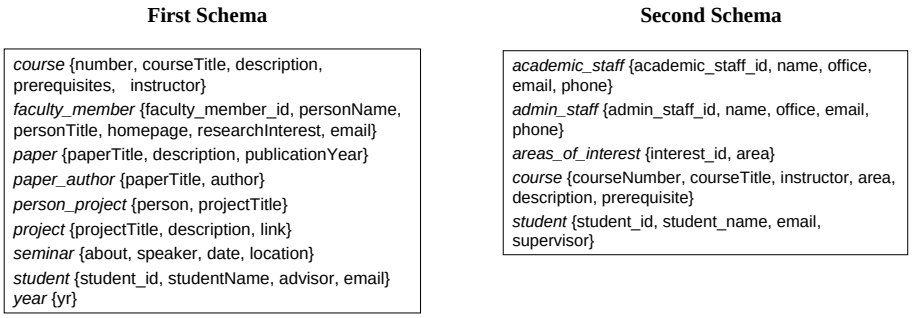


Fig. 6.16. Schema Pair#4 (UNIV-1)

Step-1: First Schema of Schema Pair#5 + Second Schema of Schema Pair#5

At the first step of schema integration test, SASMINT system has integrated two schemas of the Schema Pair#5, shown in Figure 6.14, resulting in the integrated schema, elements of which are shown in Figure 6.17. During the integration process, one redundancy was automatically generated, which was the "UNIVERSITY_REF" column of the "department" table. Therefore, the result of minimality measure was 0.99, which is a substantial automated achievement. When key minimality is considered, one redundant foreign key relationship was defined on the same "UNIVERSITY_REF" column, which resulted in a key minimality of 97%. Although the resulting integrated schema had one redundant element and foreign key, it covered all the elements and keys of two source schemas. Therefore, the result is considered as 100% complete and 100% key complete, which is again a substantial automated achievement. Further details of this step are provided in Appendix F.

Step-2: Integrated Schema#1 + First Schema of Schema Pair#6

At this step, SASMINT integrated the Integrated Schema#1 and the first schema of the Schema Pair#6, generating the Integrated Schema#2. Figure 6.18 shows only newly added tables and those tables that had changes in their columns. Due to the redundant "UNIVERSITY_REF" column and the foreign key defined on it, the result of minimality measure was 0.99 and the key minimality measure was 0.97. However, since all the concepts and keys of the first three schemas integrated (first schema of the Schema Pair#5, second schema of the Schema Pair#5, and first schema of the Schema Pair#6) were covered in the integrated schema, completeness and key completeness were again 100%.

INTEGRATED_1:university {university_ID (PK), UNIVERSITY_NAME, UNIVERSITY_ESTABLISHMENT_DATE, UNIVERSITY_WEBSITE}

INTEGRATED_1:program{program_ID (PK), PROGRAM_NAME, PROGRAM_DESC}

INTEGRATED_1:*academic_programme*{academic_programme_ID (PK), ACADEMIC_YEAR, ACADEMIC_SEMESTER, PROGRAM_REF}

INTEGRATED_1:department{department_ID (PK), DEPT_NAME, UNIVERSITY_REF(FK), FACULTY_REF(FK)}

INTEGRATED_1:course{course_ID (PK), COURSE_NAME, COURSE_CREDITS, COURSE_PROVIDER (FK), COURSE_INSTRUCTOR(FK)}

INTEGRATED_1:academic_staff_member{academic_staff_member_ID (PK), STAFF_NAME, STAFF_IDENTIFICATION_NUM, STAFF_FAX, STAFF_BIRTHDATE, OFFICE_ADDRESS, STAFF_EMAIL, STAFF_PHONE}

INTEGRATED_1:campus{campus_ID (PK), CAMPUS_NAME, CAMPUS_LOCATION, UNVCAMPUS (FK)}

INTEGRATED_1:faculty{faculty_ID (PK), FACULTY_NAME, DEAN_REF(FK), UNIVERSITY_REF (FK)}

INTEGRATED_1:registration{registration_ID (PK), REGISTRATION_ACADEMICSTAFFMEMBER_REF(FK), REGISTRATION_COURSE_REF(FK), REGISTRATION_ACADEMICPROGRAMME_REF(FK)}

INTEGRATED_1:university_student{university_student_ID (PK), NAME, ELECTRONIC_MAIL, TELEPHONE}

Fig. 6.17. Elements of Integrated Schema#1

INTEGRATED_2:payrate{Rank (PK), HrRate}

INTEGRATED_2:workson{Name, Proj, Hrs, ProjRank (FK)}

INTEGRATED_2:address{Id (PK), Street, City, PostalCode}

INTEGRATED_2:academic_staff_member{academic_staff_member_ID (PK), STAFF_NAME, STAFF_IDENTIFICATION_NUM, STAFF_FAX, STAFF_BIRTHDATE, STAFF_EMAIL, STAFF_PHONE, Sal, addr(FK)}

INTEGRATED_2:university_student{university_student_ID (PK), NAME, ELECTRONIC_MAIL, TELEPHONE, GPA, Yr}

Fig. 6.18. New Elements of Integrated Schema#2

Step-3: Integrated Schema#2 + Second Schema of Schema Pair#6

At Step-3, SASMINT generated Integrated Schema#3, by integrating the Integrated Schema#2 and the second schema of the Schema Pair#6. The only change in the new integrated schema was the addition of one new column, called "Expenses" to the "workson" table. Due to the redundant "UNIVERSITY_REF" column and the foreign key defined on it, the resulting schema was again 99% minimal and 97% key minimal. However, it was again 100% complete considering both the concepts and keys.

Step-4: Integrated Schema#3 + First Schema of Schema Pair#4

In Step-4, SASMINT integrated the Integrated Schema#3 and the first schema of the Schema Pair#4, resulting in the Integrated Schema#4. Figure 6.19 shows only the newly added tables and those tables that had changes in their columns at this step. Minimality and key minimality were 0.99 and 0.98 respectively, because of the redundant "UNIVERSITY_REF" column and the foreign key. Considering the concepts, schema was 100% complete, but since three foreign keys were missed, as explained in Appendix F, the key completeness was 0.95 after this step.

INTEGRATED_4:academic_staff_member{academic_staff_member_ID (PK), STAFF_NAME, STAFF_IDENTIFICATION_NUM, STAFF_FAX, STAFF_BIRTHDATE, STAFF_EMAIL, STAFF_PHONE, Sal, addr(FK) personTitle, homepage, researchInterest}

INTEGRATED_4:university_student{university_student_ID (PK), NAME, ELECTRONIC_MAIL, TELEPHONE, GPA, Yr, advisor(FK)}

INTEGRATED_4:paper{paperTitle (PK), description, publicationYear(FK)}

INTEGRATED_4:paper_author{paperTitle(PK)(FK), author(PK)(FK) }

INTEGRATED_4:person_project{person (PK)(FK), projectTitle (PK)(FK)}

INTEGRATED_4:seminar{about (PK), speaker (FK), location, date}

INTEGRATED_4:year{yr (PK)}

INTEGRATED_4:project{projectTitle (PK), description, link}

INTEGRATED_4:workson{Name, Hrs, ProjRank, Expenses}

Fig. 6.19. New Elements of Integrated Schema#4

Step-5: Integrated Schema#4 + Second Schema of Schema Pair#4

In the final step of schema integration, SASMINT integrated the Integrated Schema#4 and the second schema of the Schema Pair#4. Final integrated schema is called Integrated Schema#5. Figure 6.20 shows the elements of the final integrated schema. This schema was 99% minimal and 99% key minimal. Redundancy was again due to the "UNIVERSITY_REF" column and the foreign key defined on it. Although all the concepts of six schemas integrated were covered in the final schema, resulting in 100% completeness, two more foreign keys were

INTEGRATED_5:university {university_ID (PK), UNIVERSITY_NAME, UNIVERSITY_ESTABLISHMENT_DATE, UNIVERSITY_WEBSITE}

INTEGRATED_5:program{program_ID (PK), PROGRAM_NAME, PROGRAM_DESC}

INTEGRATED_5:*academic_programme*{academic_programme_ID (PK), ACADEMIC_YEAR, ACADEMIC_SEMESTER, PROGRAM_REF}

INTEGRATED_5:department{department_ID (PK), DEPT_NAME, UNIVERSITY_REF(FK), FACULTY_REF(FK)}

INTEGRATED_5:course{course_ID (PK), COURSE_NAME, COURSE_CREDITS, COURSE_PROVIDER (FK), COURSE_INSTRUCTOR(FK), description, prerequisites, area}

INTEGRATED_5:campus{campus_ID (PK), CAMPUS_NAME, CAMPUS_LOCATION, UNVCAMPUS (FK)}

INTEGRATED_5:faculty{faculty_ID (PK), FACULTY_NAME, DEAN_REF(FK), UNIVERSITY_REF (FK)}

INTEGRATED_5:registration{registration_ID (PK), REGISTRATION_ACADEMICSTAFFMEMBER_REF(FK), REGISTRATION_COURSE_REF(FK), REGISTRATION_ACADEMICPROGRAMME_REF(FK)}

INTEGRATED_5:payrate{Rank (PK), HrRate}

INTEGRATED_5:workson{Name, Hrs, ProjRank, Expenses}

INTEGRATED_5:address{Id (PK), Street, City, PostalCode}

INTEGRATED_5:academic_staff_member{academic_staff_member_ID (PK), STAFF_NAME, STAFF_IDENTIFICATION_NUM, STAFF_FAX, STAFF_BIRTHDATE, STAFF_EMAIL, STAFF_PHONE, Sal, addr(FK) personTitle, homepage}

INTEGRATED_5:university_student{university_student_ID (PK), NAME, ELECTRONIC_MAIL, TELEPHONE, GPA, Yr, advisor(FK)}

INTEGRATED_5:paper{paperTitle (PK), description, publicationYear(FK)}

INTEGRATED_5:paper_author{paperTitle (PK)(FK), author (PK)(FK) }

INTEGRATED_5:person_project{person (PK)(FK), projectTitle (PK)(FK)}

INTEGRATED_5:seminar{about (PK), speaker (FK), location, date}

INTEGRATED_5:year{yr (PK)}

INTEGRATED_5:project{projectTitle (PK), description, link}

INTEGRATED_5:admin_staff{admin_staff_id (PK), name, email, phone}

INTEGRATED_5:areas_of_interest{interest_id (PK), area (PK)}

Fig. 6.20. Elements of the **Final Integrated Schema**

missed in this step, in addition to the ones in the previous step. Therefore, the key completeness was 0.93, as explained in detail in Appendix F.

6.8 Conclusions

This chapter presents the results of our evaluation of the SASMINT system. In this chapter, first the state of the art in the schema matching evaluations is addressed, and then the quality measures that were applied during our experiments are explained. After that, the set of six test schemas that were used for evaluating the SASMINT system are introduced. Since there was not any benchmark for relational schema matching systems, we generated our own test schemas, a number of which were the same or modified versions of schemas from the evaluations of similar matching systems in related research.

After the introductory part, the results of our experiments are presented in this chapter. Schema matching in SASMINT was compared against one leading state of the art schema matching system, the COMA++. A brief summary of this comparison based on the input, the combination of matchers, the output, the persistence store, and the quality criteria is given below:

- *Input:* SASMINT accepts relational schemas, bearing in mind that most data are still stored in relational databases and corresponding schemas are represented as relational DDLs. As stated in Chapter 7 about the Future Steps, it may be possible to extend SASMINT to also support matching of XML Schema. The COMA++ accepts relational schema, XML Schema, and OWL as input to its matching procedure. In addition to the schemas to be matched, SASMINT uses a number of auxiliary inputs. A file consisting of a number of well-known abbreviations is exploited. Users can update (extend) this file with other abbreviations from the domain of schemas. As the second auxiliary input, SASMINT uses the WordNet for identifying semantic relationships between schema elements. Similar to SASMINT, COMA++ also utilizes a user-modifiable list of abbreviations. On the other hand, in order to detect synonymy relationships, COMA++ requires a user-provided list of synonyms. The disadvantage of this approach is that users are required to continuously update this list with pairs of synonyms from the domain of schemas.

- *Combination of Matchers:* SASMINT and COMA++ both provide a library of matchers. SASMINT provides the possibility of user assigned weights to different algorithms and a Sampler component, which helps the user to identify the appropriate weight for each linguistic matching metric. On the other hand, COMA++ supports different alternatives for combining, aggregating, and selecting match results from different metrics. But the user should decide and select the approaches to be applied. This feature makes it difficult for an inexperienced user to identify the best combination.

- *Output:* The output of a match system is a mapping, indicating which elements of the recipient and donor schemas correspond to each other. Both SASMINT and COMA++ represent these correspondences using a value between 0 and 1. Furthermore, they both can support 1-to-1, 1-to-n, n-to-1, and m-to-n types of matches.

- *Persistence store for the results:* For matching and integration of schemas, SASMINT stores the results based on SDML. This allows the results to be used for federated query processing and for decomposition of queries to be sent to different local schemas, as well as for formal representation of the semi-automatically generated integrated schema from the recipient and donor schemas. COMA++ has an internal repository for the results, but users cannot see in which format results are stored and it is not clear how to use these results outside of the system.

- *Quality of Schema Matching*: The quality of schema matching supported by SASMINT and COMA++ was compared using their default settings for the combination of different matchers. SASMINT's default approach for combining linguistic and structure matching metrics calculates their weighted sum. However, then the Linguistic metrics have a higher impact (0.7) than the structure ones (0.3), on the final result. But in the evaluation between the two systems, each metric in groups of the linguistic matching and structure matching was considered with equal weight. Namely, in order to make a fair comparison with COMA++, we did not give higher weights to the metrics that could be more appropriate for some schema types. COMA matcher combines name, path, leaves, parents, and siblings matchers by averaging them. We updated the abbreviation lists of both systems with new abbreviations related to schemas. However, we did not update the synonyms list of COMA++, because manually adding into this list some complex semantic correspondences would also lead to unfair comparisons. We carried out experiments based on two types of result selection strategies that we call as: 1) Select all above threshold and 2) select max above threshold. Both systems performed better in the second approach. When the first approach was used, results for COMA++ were worse than those of SASMINT. For the second approach, the systems performed the same for some schema pairs, for the remaining pairs, SASMINT performed better than COMA++.

In order to evaluate the Sampler component of SASMINT, we performed some tests using the same set of test schemas. For this purpose, after setting the threshold value, we provided the Sampler component with a number of similar pairs from the two schemas being compared. We performed schema matching using both the Sampler's computed weights as well as the equal weights for linguistic matching algorithms. In some cases, using Sampler's computed weights resulted in an increase in the number of false positives, and thus a decrease in the precision. However, in every such case since Sampler identifies higher number of correct matches, by assigning appropriate weights, the corresponding recall was much better than the case where Sampler was not used. Therefore, even in these cases, this resulted in an increase in f-measure and overall performance of SASMINT. Therefore, using Sampler was shown to improve the quality of match results.

After evaluating the schema matching approach of SASMINT against the leading system COMA++, we evaluated the schema integration approach of SASMINT. Since COMA++'s schema merging feature is very primitive and there were no other systems at the level of SASMINT, which can use their schema matching results for semi-automatic schema integration, we could unfortunately not compare the results of schema integration approach of SASMINT with any other system. Nevertheless, we performed the incremental integration of six schemas to be able to evaluate SASMINT against the state of the art criteria defined for automated schema integration. During the empirical evaluation, SASMINT achieved a high percentage of minimality and completeness for its integrated schemas procedure, which applies its user-validated matches.

To sum up, schema matching and schema integration are two challenging tasks in SASMINT. Different types of schema heterogeneities, such as semantic and structural, make these tasks more difficult to achieve automatically. A semi-automatic system might perform badly on such schemas. Evaluation data sets need to be carefully selected to cover different types of schema heterogeneities. Furthermore, in order to fairly evaluate the schema matching and schema integration systems, measures need to be carefully selected and defined to consider all aspects of a system in evaluation, such as quality of the match and integration results, how the results are represented, how easily these results can be modified/corrected by the user, and whether it is possible to use these results in other processes like query decomposition in federated query processing.

Chapter 7

Thesis conclusions and future work

7.1 Summary of General Approach

The importance of developing a supporting infrastructure for data sharing has been understood clearly during the last years, with the increasing need for collaboration among organizations in a wide variety of domains, from manufacturing and service industry to scientific virtual laboratory and disaster management. In order to facilitate and enable collaboration among distributed, heterogeneous, and autonomous organizations, one of the first requirements that needs to be met is enabling access to certain data that is to be shared among the stakeholder organizations. However, before any sharing of data could possibly occur, many existing syntactic, semantic, and structural heterogeneities among the stakeholder database schemas need to be resolved. Manual resolution of schema heterogeneities is very time consuming, cumbersome, and error prone. This becomes more challenging when scaling up is required in large networks. Namely, without automated ways of removing such heterogeneities between separate database schemas of participants, data interoperability and therefore effective collaboration goals cannot be met.

Consequently, provision of automated schema matching and integration tools is an active area of research with numerous technical challenges. One of the biggest challenges in this area is the automatic **resolution of database schema heterogeneity**, without which provision of integrated data access and sharing among autonomous, heterogeneous, and distributed databases will remain difficult to achieve.

In this thesis, we propose a supervised automated approach to solve both the problem of schema matching and the schema integration assisting the users with removal of heterogeneities among source database schemas and to integrate them effectively. We also provide an implementation of this approach in the form of a software system, which we call the Semi-Automatic Schema Matching and INTegration (SASMINT).

As the **first** step towards the provision of a supporting infrastructure for data sharing in collaborative networks of organizations, we have performed an analysis of different types of information sharing heterogeneities. Furthermore, we have identified the varieties of heterogeneities that represent the most important obstacles to Schema Matching and Schema Integration tasks.

As the **second** step, we have analyzed related research on Schema Matching and Schema Integration approaches. Based on this survey and the identified open issues, we have devised

our approach to deal with many challenges, and have proposed a new approach for schema matching and schema integration in relational databases. In order to not reinvent the wheel, we have combined a number of well-known algorithms suggested in related research tackling some of the challenges of matching terms from different domains. We have generated an innovative mechanism and format to represent the results generated by the schema matching and schema integration processes. Furthermore, we have proposed an approach for automatically generating an integrated schema using the results of schema matching through the design and development of a number of heuristic rules. Finally, we have defined a derivation language to formally specify and store how the integrated schema is derived from its input donor and recipient schemas.

As the **third** step, we have implemented our approach to semi-automatic schema matching and schema integration both as a proof of concept and in order to verify and validate it. The main components of the SASMINT system architecture, which are implemented in this thesis comprise:

a) *Sampler Component*, which helps users with automatic identification of appropriate weight for each algorithm used for linguistic matching.
b) *Graph Representation Component*, which is responsible for representing schemas in the DAG format.
c) *GUI Component*, which enables users to interact with the system to configure needed parameters and to modify and accept the results generated by schema matching and schema integration processes.
d) *Schema Matching Component*, which matches the recipient and the donor schemas using a combination of linguistic and structure matching techniques.
e) *Schema Integration Component*, which both integrates the donor and recipient schemas using the set of pre-defined rules and generates the formal specification of integrated schema results using a derivation language.

As the **fourth** step, we have finally evaluated our approach for Schema Matching in comparison with the most closely related approach and system, the COMA++'s approach, through experimenting with six pairs of schemas consisting of a variety of heterogeneities. Furthermore, we have also evaluated our schema integration approach. We could not compare this part against other schema integration approaches, since there was no other system similar to SASMINT that uses its results from schema matching for the purpose of semi-automatic schema integration. Our experiment results for schema matching and schema integration have shown that SASMINT provides good quality results and higher than its closely related competitor.

7.2 Reflections on the Research Questions

> **RQ1.** Which effective approaches and architectures can enable data sharing through interlinking and/or integrating heterogeneous databases of distributed nodes?

Before establishing a solution for a problem, it is important to understand all concepts and terminology related to it. In order to meet this requirement, in Chapter 2, we provided definitions of a variety of terms in the database management research domain concerning approaches, architectures, and systems for interlinking and/or integrating heterogeneous data provided by distributed nodes, in order to enable data sharing among them. Since in the research literature quite often the same term was used to mean different things and different terms were used to refer to the same concept, we thought that it was crucial to differentiate

among these definitions and clarify the terminology used in the research work explained in this thesis. For this purpose, in Chapter 2, we described a number of concepts related to distributed information management and we provided our classification for multidatabases, based on schema coupling. We then defined schema matching and schema integration and specified how they relate to distributed information management.

> **RQ2.** What is a representative taxonomy for addressing database schema heterogeneities, and in turn applicable to formalization of schema matching and schema integration challenges?

Heterogeneity is the biggest obstacle to schema matching and schema integration. In Chapter 3, we presented different types of heterogeneities that exist among information systems. Information systems heterogeneity ranges from the heterogeneity of information and its definition and classification, to the systems heterogeneity. Considering the aim of schema matching and schema integration processes, schema conflicts are the ones that need to be tackled. We categorize schema conflicts as structural and linguistic and the linguistic conflicts further as syntactic and semantic. Especially semantic and structural conflicts are difficult to automatically resolve. The more a schema matching and integration approach can automatically resolve such conflicts, the higher the value of the approach, as less user input is required.

> **RQ3.** What are effective mechanisms for semi-automatic schema matching and schema integration, and how should the user be involved in the process?

In Chapter 4, we addressed a number of efforts and systems related to providing access to heterogeneous databases. We examined them in four main groups: 1) database integration and interoperability approaches, 2) schema matching approaches, 3) schema integration approaches, and 4) ontology matching and merging approaches. Database integration and interoperability approaches typically do not consider any automation in schema matching. Schema matching approaches, on the other hand, are either limited in the solutions that they provide or utilize a few match algorithms resolving only specific heterogeneities. They still require a lot of manual input. Furthermore, most of these schema matching approaches do not provide any GUI for helping users to modify match results and do not address using their results for schema integration. While very much related, the schema matching is seen as a separate problem than schema integration, and using the match results for semi-automatic schema integration is not taken into account. As for the schema integration approaches, their provided solutions are not generic enough. They generally assume that correspondences among schemas are already a given input. We proposed the SASMINT approach to overcome the limitations of previous approaches. It supports both semi-automatic schema matching and schema integration. SASMINT addresses and handles all types of conflicts addressed in Section 3.3. However, a fully automatic resolution is not possible for some types of semantic and structural conflicts, as described earlier and thus user input might be required in some cases. SASMINT uses a combination of linguistic and structure matching metrics and algorithms in order to resolve different types of conflicts addressed in Section 3.3. A novel way of identifying appropriate weights for each metric and algorithm is also proposed. It also represent that once formally specified, the results of schema matching can be exploited for semi-automatic schema integration. By means of a GUI, users can easily modify and store the match and integration results. SASMINT defines an XML-based derivation language as the storage format for the results of matching and integration. In order to verify and validate the SASMINT approach, we have implemented it. In Chapter 5, we provided details of the development architecture of SASMINT together with a number of screenshots of this system.

> **RQ4.** How can we assess and validate the effectiveness of the proposed semi-automatic approaches for schema matching and schema integration?

In order to validate the proposed approach, its evaluation needs to be done against the leading competitors and/or the well defined generic measures. In Chapter 6, we explained how we evaluated the approach of SASMINT. In order to measure the quality of schema matching, a number of well-known measures are addressed, namely, the precision, recall, f-measure, and overall. Then, SASMINT is compared applying these measures, against a leading competitor. For the schema integration, completeness and minimality measures are introduced and applied to identify how well the input schemas are combined by the integrated schema approach and whether the integrated schema is optimal or if it contains redundant elements or keys. After identifying the set of test schemas, quality of schema matching approach of SASMINT was compared to that of COMA++. It was shown that SASMINT was at least as good as if not better than one of the leading state of the art schema matching systems. Evaluation of schema integration experiment of SASMINT also generated very promising results. Furthermore, we also performed some tests to evaluate the Sampler component. In these tests, we identified that Sampler helped to improve the quality of the match results.

7.3 Future Work

There are several areas of research that can continue and further extend certain aspects and features of the work presented in this thesis:

- *Support for XML Schema*

Considering the current extensive use of relational databases, the implementation of the proposed SASMINT system that is provided in this thesis supports matching and integration of relational schemas only. However, both the system and the data architecture of SASMINT have been designed to also be able to support matching and integration of other frequently used data model representations, such as the XML Schema. For supporting matching of XML Schemas, the adapter framework that is now in place needs to be extended with XML Schema import features for incorporating XML Schema support features. In order to integrate XML Schemas, new integration rules need to be defined into SASMINT.

- *Support for Ontology*

Similar to what is stated above for XML Schema support, SASMINT could be extended with the support for Ontologies. This extension would require more work compared to XML Schema support, since the semantics of Ontologies could entail incorporation of technologies/tools like inference/reasoning engines.

- *Using Machine Learning Techniques for the Sampler Component*

At present Sampler applies an approach based on f-measure to identify the best applicable weights for each linguistic matching algorithm in relation to the considered specific schemas. However, it could be further extended to also utilize machine learning techniques for this purpose. Machine learning algorithms can examine large amounts of data and make intelligent decisions based on these data. Therefore, by learning from the true and false positive matches, machine learning algorithms might identify more appropriate weights.

- *Creating a benchmark for schema matching and integration*

In our evaluation studies, one challenge was to find/design objective (relational) schemas that we could use to measure the functional performance of our schema matching and integration system. The test schemas used by other schema matching and evaluation research were designed for a specific need in mind and consisted of only certain types of schema heterogeneities. As a future work, the creation of a benchmark would be valuable, in order to generate more generic schemas to serve as the base for comparable evaluation between systems.

- *Fragmented Matching and Integration*

Current focus of SASMINT is to address and resolve different types of heterogeneities, when two schemas are compared, and not addressing very large schemas. However, future work could consider matching and integrating very large schemas. This would require enabling the fragmented matching and integration in order to make it easier to compare and integrate big schemas, and in turn require the identification of most appropriate fragments for this purpose.

Appendix A

List of author's publications

Journal Articles

1. Unal, O., & Afsarmanesh, H. (2010). Semi-automated schema integration with SASMINT. *Journal of Knowledge and Information Systems, 23(1)*, ISSN: 0219-1377, pp. 99-128.
2. Unal, O., & Afsarmanesh, H. (2009). Schema matching and integration for data sharing among collaborating organizations. *Journal of Software, 4*(3), ISSN:1796-217X, pp.248-261.
3. Unal, O., & Afsarmanesh, H. (2006). SASMINT System for Database Interoperability in Collaborative Networks. *Lecture Notes in Computer Science* (LNCS 4275), ISBN: 978-3-540-48287-3, Springer Berlin/Heidelberg, pp. 91-108.

Book Chapters

1. Unal, O., Kaletas, E. C., Afsarmanesh, H., Yakali, H. H., & Hertzberger, L. O. (2005). Collaborative information management system for science domains. In S. Dasgupta (Ed.), *Encyclopedia of virtual communities and technologies*. Idea Group Publishing.

Peer Reviewed International Conference Papers

1. Unal, O., & Afsarmanesh, H. (2006). Interoperability in collaborative network of biodiversity organizations. In: *Proceedings of the PRO-VE - Network-Centric Collaboration and Supporting Frameworks*, Helsinki, Finland. Springer, pp.515-524.

2. Unal, O., & Afsarmanesh, H. (2006). Using linguistic techniques for schema matching. In: *Proceedings of the International Conference on Software and Data Technologies (ICSOFT)*, Setubal, Portugal. INSTICC Press, pp.115-120, ISBN:972-8865-69-4.

3. Guevara-Masis, V., Unal, O., Kaletas, E. C., Afsarmanesh, H., & Hertzberger, L. O. (2004). Using ontologies for collaborative information management: Some challenges & ideas. In: *Proceedings of the Third Biennial International Conference on Advances in Information Systems (ADVIS),* Izmir, Turkey. Springer Lecture Notes in Computer Science (LNCS 3261), pp.107-116, ISBN:3-540-23478-0.

4. Garita, C., Afsarmanesh, H., Unal, O., Hertzberger, L.O. (2003). Building a virtual laboratory for scientific experimentation in molecular biology. In: *Proceedings of PRO-VE'03 - Processes and Foundations for Virtual Organizations*, Kluwer Academic Publishers, pp. 181-190.

Appendix B

XSD for SDML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://namespaces.sasmint.org/2007/04/GraphModel"
targetNamespace="http://namespaces.sasmint.org/2007/04/GraphModel" elementFormDefault="qualified"
attributeFormDefault="qualified" version="0.9">
	<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>
	<xs:element name="columnRenameDerivation">
		<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="columnStringAdditionDerivation">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="ns1:derivationNode" maxOccurs="unbounded"/>
				<xs:element ref="ns1:derivationType" minOccurs="0"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="columnUnionDerivation">
```

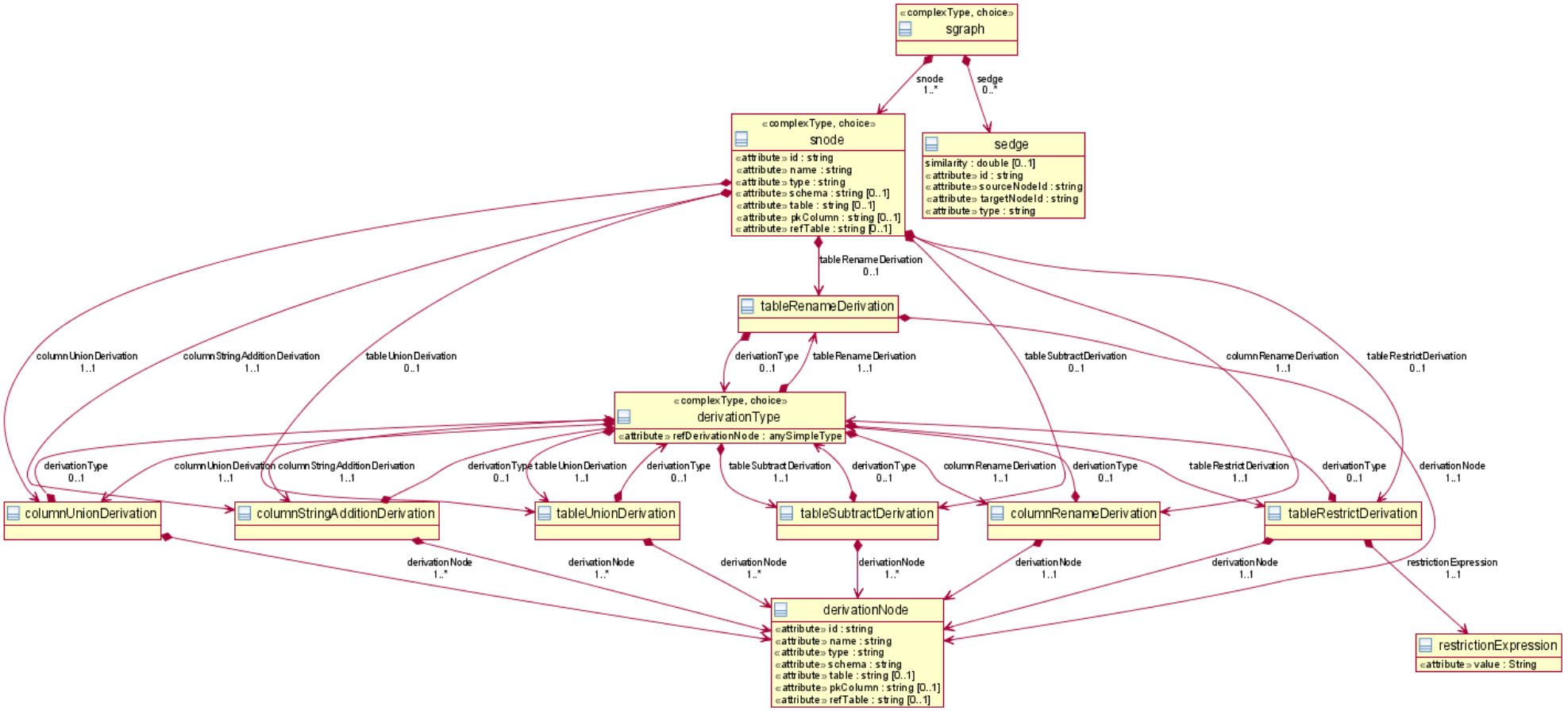
```
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="ns1:derivationNode" maxOccurs="unbounded"/>
                    <xs:element ref="ns1:derivationType" minOccurs="0"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <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="sedge">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="ns1:similarity" minOccurs="0"/>
                </xs:sequence>
                <xs:attribute name="id" type="xs:string" use="required"/>
                <xs:attribute name="sourceNodeId" type="xs:string" use="required"/>
                <xs:attribute name="targetNodeId" type="xs:string" use="required"/>
                <xs:attribute name="type" type="xs:string" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="sgraph">
            <xs:complexType>
                <xs:choice maxOccurs="unbounded">
                    <xs:element ref="ns1:snode" maxOccurs="unbounded"/>
                    <xs:element ref="ns1:sedge" minOccurs="0"/>
                </xs:choice>
            </xs:complexType>
        </xs:element>
        <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="restrictionExpression">
            <xs:complexType>
```

```
                    <xs:attribute name="value" use="required">
                        <xs:simpleType>
                            <xs:restriction base="xs:string"/>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
            <xs:element name="similarity" type="xs:double"/>
            <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="tableRestrictDerivation">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="ns1:derivationNode"/>
                        <xs:element ref="ns1:restrictionExpression"/>
                        <xs:element ref="ns1:derivationType" minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="tableSubtractDerivation">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="ns1:derivationNode" maxOccurs="unbounded"/>
                        <xs:element ref="ns1:derivationType" minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="tableUnionDerivation">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="ns1:derivationNode" maxOccurs="unbounded"/>
                        <xs:element ref="ns1:derivationType" minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
</xs:schema>
```

Appendix C

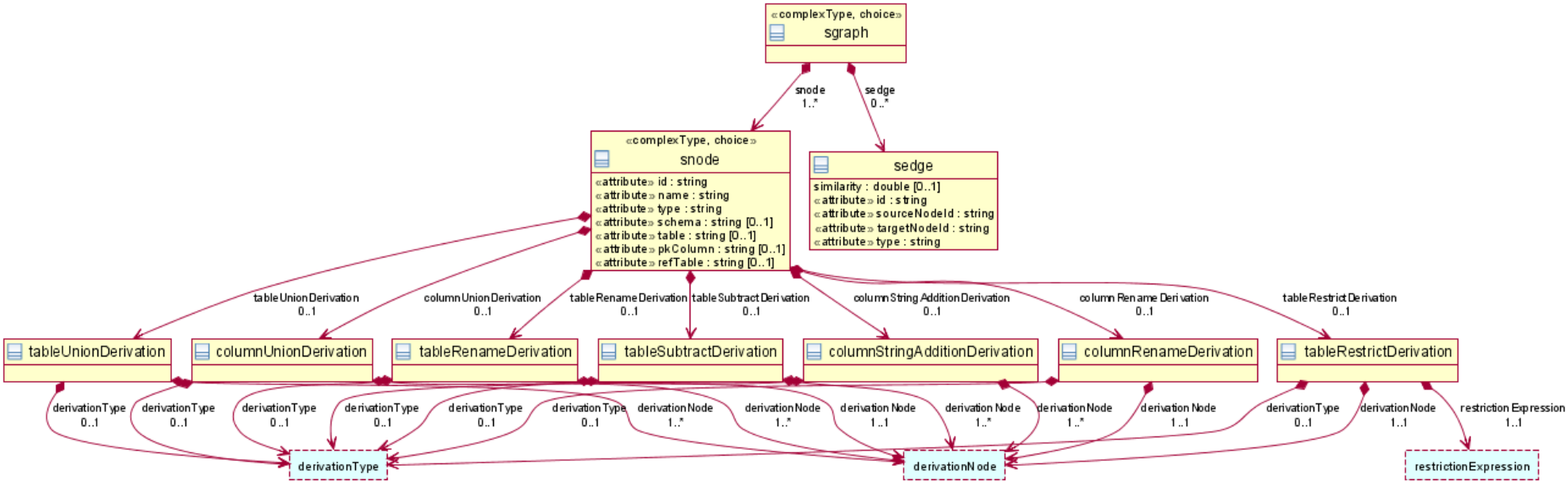
Class diagram for SDML

SDML class diagram of SASMINT



Partial SDML class diagram

(some details dropped to create better visibility)



Appendix D

Test schemas

Purchase Order Schemas (PO)

Recipient Schema

```
CREATE TABLE `customer` (
  `custNo` int(10) unsigned NOT NULL,
  `name` varchar(70) NOT NULL,
  `city` varchar(45) NOT NULL,
  `street` varchar(50) NOT NULL,
  `zip` varchar(10) NOT NULL,
  `telephone` varchar(15) NOT NULL,
  PRIMARY KEY  (`custNo`)
)

CREATE TABLE `product` (
  `productNo` int(10) unsigned NOT NULL,
  `productName` varchar(45) NOT NULL,
  `price` varchar(45) NOT NULL,
  `stock` varchar(45) NOT NULL,
  `supplierNo` int(10) unsigned NOT NULL,
  PRIMARY KEY  (`productNo`),
  KEY `FK_product_1` (`supplierNo`),
  CONSTRAINT `FK_product_1` FOREIGN KEY
(`supplierNo`) REFERENCES `supplier`
(`supplierNo`)
)

CREATE TABLE `purchase_order` (
  `purchaseOrderNo` int(10) unsigned NOT NULL,
  `custNo` int(10) unsigned NOT NULL,
  `purchaseOrderDate` datetime NOT NULL,
```

Donor Schema

```
CREATE TABLE `buyer` (
  `buyer_No` int(10) unsigned NOT NULL,
  `f_name` varchar(45) NOT NULL,
  `l_name` varchar(45) NOT NULL,
  `phone` varchar(45) NOT NULL,
  `buyer_street` varchar(45) NOT NULL,
  `buyer_city` varchar(45) NOT NULL,
  `buyer_zip` varchar(45) NOT NULL,
  PRIMARY KEY  (`buyer_No`)
)

CREATE TABLE `item` (
  `item_no` int(10) unsigned NOT NULL,
  `item_name` varchar(45) NOT NULL,
  `cost` double NOT NULL,
  `stock` varchar(45) NOT NULL,
  `item_arrival_date` datetime NOT NULL,
  `item_color_id` int(10) unsigned NOT NULL,
  `item_size_id` int(10) unsigned NOT NULL,
  PRIMARY KEY  (`item_no`),
  KEY `FK_item_1` (`item_color_id`),
  KEY `FK_item_2` (`item_size_id`),
  CONSTRAINT `FK_item_1` FOREIGN KEY
(`item_color_id`) REFERENCES `item_color`
(`item_color_id`),
  CONSTRAINT `FK_item_2` FOREIGN KEY
(`item_size_id`) REFERENCES `item_size`
```

```
  `status` varchar(45) NOT NULL,
  `deliverDate` datetime NOT NULL,
  `deliverCity` varchar(45) NOT NULL,
  `deliverStreet` varchar(50) NOT NULL,
  `deliverZip` varchar(10) NOT NULL,
  PRIMARY KEY  (`purchaseOrderNo`),
  KEY `FK_purchase_order_1` (`custNo`),
  CONSTRAINT `FK_purchase_order_1` FOREIGN
KEY (`custNo`) REFERENCES `customer` (`custNo`)
)

CREATE TABLE `purchase_order_line` (
  `purchaseOrderLineNo` int(10) unsigned NOT
NULL,
  `purchaseOrderNo` int(10) unsigned NOT NULL,
  `productNo` int(10) unsigned NOT NULL,
  `quantity` int(10) unsigned NOT NULL,
  `deliverDate` datetime NOT NULL,
  PRIMARY KEY  (`purchaseOrderLineNo`),
  KEY `FK_purchase_order_line_1`
(`purchaseOrderNo`),
  KEY `FK_purchase_order_line_2` (`productNo`),
  CONSTRAINT `FK_purchase_order_line_1`
FOREIGN KEY (`purchaseOrderNo`) REFERENCES
`purchase_order` (`purchaseOrderNo`),
  CONSTRAINT `FK_purchase_order_line_2`
FOREIGN KEY (`productNo`) REFERENCES
`product` (`productNo`)
)

CREATE TABLE `supplier` (
  `supplierNo` int(10) unsigned NOT NULL,
  `supplierName` varchar(70) NOT NULL,
  `supplierAddress` varchar(80) NOT NULL,
  PRIMARY KEY  (`supplierNo`)
)
```

```
(`item_size_id`)
)

CREATE TABLE `item_color` (
  `item_color_id` int(10) unsigned NOT NULL,
  `color_description` varchar(45) NOT NULL,
  PRIMARY KEY  (`item_color_id`)
)

CREATE TABLE `item_size` (
  `item_size_id` int(10) unsigned NOT NULL,
  `size_description` varchar(45) NOT NULL,
  PRIMARY KEY  (`item_size_id`)
)

CREATE TABLE `po` (
  `po_no` int(10) unsigned NOT NULL,
  `buyer_no` int(10) unsigned NOT NULL,
  `item_no` int(10) unsigned NOT NULL,
  `deliver_street` varchar(45) NOT NULL,
  `deliver_city` varchar(45) NOT NULL,
  `deliver_zip` varchar(10) NOT NULL,
  `deliver_date` datetime NOT NULL,
  PRIMARY KEY  (`po_no`),
  KEY `FK_po_1` (`buyer_no`),
  KEY `FK_po_2` (`item_no`),
  CONSTRAINT `FK_po_1` FOREIGN KEY
(`buyer_no`) REFERENCES `buyer` (`buyer_No`),
  CONSTRAINT `FK_po_2` FOREIGN KEY
(`item_no`) REFERENCES `item` (`item_no`)
)
```

Hotel Schemas (HOTEL)

Recipient Schema

```
CREATE TABLE `num_beds` (
  `numBedsID` varchar(50) NOT NULL,
  `numBedsAttrib` varchar(50) default NULL,
  PRIMARY KEY  (`numBedsID`)
)

CREATE TABLE `one_room` (
  `oneRoomID` varchar(50) NOT NULL,
  `roomNum` varchar(50) default NULL,
  `hasNumBedsAttribID` varchar(50) default NULL,
  `hasOnFloorAttribID` varchar(50) default NULL,
  `hasSmokingPreferenceAttribID` varchar(50) default
NULL,
  PRIMARY KEY  (`oneRoomID`),
  KEY `hasNumBedsAttribID`
```

Donor Schema

```
CREATE TABLE `num_beds_attribute` (
  `numBedsAttributeID` varchar(50) NOT NULL,
  `numBedsAttrib` varchar(50) default NULL,
  PRIMARY KEY  (`numBedsAttributeID`)
)

CREATE TABLE `on_floor_attribute` (
  `onFloorAttributeID` varchar(50) NOT NULL,
  `onFloorAttrib` varchar(50) default NULL,
  PRIMARY KEY  (`onFloorAttributeID`)
)

CREATE TABLE `room` (
  `roomID` varchar(50) NOT NULL,
  `roomNum` varchar(50) default NULL,
```

```
(`hasNumBedsAttribID`),
  KEY `hasOnFloorAttribID` (`hasOnFloorAttribID`),
  KEY `hasSmokingPreferenceAttribID`
(`hasSmokingPreferenceAttribID`),
  CONSTRAINT `oneroom_ibfk_1` FOREIGN KEY
(`hasNumBedsAttribID`) REFERENCES `numbeds`
(`numBedsID`),
  CONSTRAINT `oneroom_ibfk_2` FOREIGN KEY
(`hasOnFloorAttribID`) REFERENCES `onfloor`
(`onFloorID`),
  CONSTRAINT `oneroom_ibfk_3` FOREIGN KEY
(`hasSmokingPreferenceAttribID`) REFERENCES
`smokingpreference` (`smokingPreferenceID`)
)

CREATE TABLE `on_floor` (
  `onFloorID` varchar(50) NOT NULL,
  `onFloorAttrib` varchar(50) default NULL,
  PRIMARY KEY  (`onFloorID`)
)

CREATE TABLE `smoking_preference` (
  `smokingPreferenceID` varchar(50) NOT NULL,
  `smokingPreferenceAttrib` varchar(50) default
NULL,
  PRIMARY KEY  (`smokingPreferenceID`)
)

CREATE TABLE `suite` (
  `suiteID` varchar(50) NOT NULL,
  `roomNum` varchar(50) default NULL,
  `hasNumBedsAttribID` varchar(50) default NULL,
  `hasOnFloorAttribID` varchar(50) default NULL,
  `hasSmokingPreferenceAttribID` varchar(50) default
NULL,
  PRIMARY KEY  (`suiteID`),
  KEY `hasNumBedsAttribID`
(`hasNumBedsAttribID`),
  KEY `hasOnFloorAttribID` (`hasOnFloorAttribID`),
  KEY `hasSmokingPreferenceAttribID`
(`hasSmokingPreferenceAttribID`),
  CONSTRAINT `suite_ibfk_1` FOREIGN KEY
(`hasNumBedsAttribID`) REFERENCES `numbeds`
(`numBedsID`),
  CONSTRAINT `suite_ibfk_2` FOREIGN KEY
(`hasOnFloorAttribID`) REFERENCES `onfloor`
(`onFloorID`),
  CONSTRAINT `suite_ibfk_3` FOREIGN KEY
(`hasSmokingPreferenceAttribID`) REFERENCES
`smokingpreference` (`smokingPreferenceID`)
)

CREATE TABLE `town_house` (
  `townHouseID` varchar(50) NOT NULL,
  `roomNum` varchar(50) default NULL,
  `hasNumBedsAttribID` varchar(50) default NULL,
  `hasOnFloorAttribID` varchar(50) default NULL,
  `hasSmokingPreferenceAttribID` varchar(50) default
NULL,
  `numBedsAttribID` varchar(50) default NULL,
  `smokingOrNoAttribID` varchar(50) default
NULL,
  `onFloorAttribID` varchar(50) default NULL,
  `sizeOfRoomAttribID` varchar(50) default NULL,
  PRIMARY KEY  (`roomID`),
  KEY `numBedsAttribID` (`numBedsAttribID`),
  KEY `smokingOrNoAttribID`
(`smokingOrNoAttribID`),
  KEY `onFloorAttribID` (`onFloorAttribID`),
  KEY `sizeOfRoomAttribID`
(`sizeOfRoomAttribID`),
  CONSTRAINT `room_ibfk_1` FOREIGN KEY
(`numBedsAttribID`) REFERENCES
`numbedsattribute` (`numBedsAttributeID`),
  CONSTRAINT `room_ibfk_2` FOREIGN KEY
(`smokingOrNoAttribID`) REFERENCES
`smokingattribute` (`smokingAttributeID`),
  CONSTRAINT `room_ibfk_3` FOREIGN KEY
(`onFloorAttribID`) REFERENCES
`onfloorattribute` (`onFloorAttributeID`),
  CONSTRAINT `room_ibfk_4` FOREIGN KEY
(`sizeOfRoomAttribID`) REFERENCES
`sizeofroomattribute` (`sizeOfRoomAttributeID`)
)

CREATE TABLE `size_of_room_attribute` (
  `sizeOfRoomAttributeID` varchar(50) NOT
NULL,
  `sizeOfRoomAttrib` varchar(50) default NULL,
  PRIMARY KEY  (`sizeOfRoomAttributeID`)
)

CREATE TABLE `smoking_attribute` (
  `smokingAttributeID` varchar(50) NOT NULL,
  `smokingAttrib` varchar(50) default NULL,
  PRIMARY KEY  (`smokingAttributeID`)
)
```

PRIMARY KEY (`townHouseID`), KEY `hasNumBedsAttribID` (`hasNumBedsAttribID`), KEY `hasOnFloorAttribID` (`hasOnFloorAttribID`), KEY `hasSmokingPreferenceAttribID` (`hasSmokingPreferenceAttribID`), CONSTRAINT `townhouse_ibfk_1` FOREIGN KEY (`hasNumBedsAttribID`) REFERENCES `numbeds` (`numBedsID`), CONSTRAINT `townhouse_ibfk_2` FOREIGN KEY (`hasOnFloorAttribID`) REFERENCES `onfloor` (`onFloorID`), CONSTRAINT `townhouse_ibfk_3` FOREIGN KEY (`hasSmokingPreferenceAttribID`) REFERENCES `smokingpreference` (`smokingPreferenceID`))	

Biology Schemas (SDB)

Recipient Schema	**Donor Schema**
CREATE TABLE `animal_donor` (`animalID` varchar(50) NOT NULL, `strain` varchar(50) default NULL, `species` varchar(50) default NULL, PRIMARY KEY (`id`), CONSTRAINT `animal_donor_ibfk_1` FOREIGN KEY (`id`) REFERENCES `donor` (`id`)) CREATE TABLE `diagnoses` (`diagID` varchar(50) NOT NULL, `donor` varchar(50) default NULL, PRIMARY KEY (`diagID`), KEY `donor` (`donor`), CONSTRAINT `diagnoses_ibfk_1` FOREIGN KEY (`donor`) REFERENCES `human_donor` (`humanID`)) CREATE TABLE `donor` (`id` varchar(50) NOT NULL, `gender` varchar(50) default NULL, `species` varchar(50) default NULL, PRIMARY KEY (`id`)) CREATE TABLE `family_history` (`histID` varchar(50) NOT NULL, `donor` varchar(50) default NULL, PRIMARY KEY (`histID`), KEY `donor` (`donor`), CONSTRAINT `family_history_ibfk_1` FOREIGN KEY (`donor`) REFERENCES `human_donor` (`id`)	CREATE TABLE `diagnoses` (`diagID` varchar(50) NOT NULL, `visitUpdate` varchar(50) default NULL, PRIMARY KEY (`diagID`), KEY `visitUpdate` (`visitUpdate`), CONSTRAINT `diagnoses_ibfk_1` FOREIGN KEY (`visitUpdate`) REFERENCES `visit_update` (`updateID`)) CREATE TABLE `donor` (`id` varchar(50) NOT NULL, `gender` varchar(50) default NULL, `species` varchar(50) default NULL, `strain` varchar(50) default NULL, `dob` varchar(50) default NULL, PRIMARY KEY (`id`)) CREATE TABLE `donor_visit` (`donor_visit_id` varchar(50) NOT NULL, `donor` varchar(50) default NULL, `content` varchar(50) default NULL, PRIMARY KEY (`id`), KEY `donor` (`donor`), CONSTRAINT `donor_visit_ibfk_1` FOREIGN KEY (`donor`) REFERENCES `donor` (`id`)) CREATE TABLE `family_history` (`histID` varchar(50) NOT NULL, `visitUpdate` varchar(50) default NULL, PRIMARY KEY (`histID`),

```
)

CREATE TABLE `human_donor` (
  `humanID` varchar(50) NOT NULL,
  `dob` varchar(50) default NULL,
  `gender` varchar(50) default NULL,
  PRIMARY KEY  (`id`),
  CONSTRAINT `human_donor_ibfk_1` FOREIGN
KEY (`humanID`) REFERENCES `donor` (`id`)
)

CREATE TABLE `lab_test` (
  `testID` varchar(50) NOT NULL,
  `donor` varchar(50) default NULL,
  PRIMARY KEY  (`testID`),
  KEY `donor` (`donor`),
  CONSTRAINT `lab_test_ibfk_1` FOREIGN KEY
(`donor`) REFERENCES `donor` (`id`)
)

CREATE TABLE `life_style_factors` (
  `factID` varchar(50) NOT NULL,
  `donor` varchar(50) default NULL,
  PRIMARY KEY  (`factID`),
  KEY `donor` (`donor`),
  CONSTRAINT `life_style_factors_ibfk_1`
FOREIGN KEY (`donor`) REFERENCES
`human_donor` (`id`)
)

CREATE TABLE `medications` (
  `medicID` varchar(50) NOT NULL,
  `donor` varchar(50) default NULL,
  PRIMARY KEY  (`medicID`),
  KEY `donor` (`donor`),
  CONSTRAINT `medications_ibfk_1` FOREIGN
KEY (`donor`) REFERENCES `human_donor` (`id`)
)

CREATE TABLE `sample` (
  `name` varchar(50) NOT NULL,
  `donorID` varchar(50) NOT NULL,
  PRIMARY KEY  (`name`),
  KEY `FK_sample_1` (`donorID`),
  CONSTRAINT `FK_sample_1` FOREIGN KEY
(`donorID`) REFERENCES `donor` (`id`)
)
```

```
  KEY `visitUpdate` (`visitUpdate`),
  CONSTRAINT `family_history_ibfk_1` FOREIGN
KEY (`visitUpdate`) REFERENCES `visit_update`
(`updateID`)
)

CREATE TABLE `lab_test` (
  `testID` varchar(50) NOT NULL,
  `visitUpdate` varchar(50) default NULL,
  PRIMARY KEY  (`testID`),
  KEY `visitUpdate` (`visitUpdate`),
  CONSTRAINT `lab_test_ibfk_1` FOREIGN KEY
(`visitUpdate`) REFERENCES `visit_update`
(`updateID`)
)

CREATE TABLE `life_style_factors` (
  `factID` varchar(50) NOT NULL,
  `visitUpdate` varchar(50) default NULL,
  PRIMARY KEY  (`factID`),
  KEY `visitUpdate` (`visitUpdate`),
  CONSTRAINT `life_style_factors_ibfk_1`
FOREIGN KEY (`visitUpdate`) REFERENCES
`visit_update` (`updateID`)
)

CREATE TABLE `medications` (
  `medicID` varchar(50) NOT NULL,
  `visitUpdate` varchar(50) default NULL,
  PRIMARY KEY  (`medicID`),
  KEY `visitUpdate` (`visitUpdate`),
  CONSTRAINT `medications_ibfk_1` FOREIGN
KEY (`visitUpdate`) REFERENCES `visit_update`
(`updateID`)
)

CREATE TABLE `sample` (
  `name` varchar(50) NOT NULL,
  `donorVisitID` varchar(50) default NULL,
  PRIMARY KEY  (`name`),
  KEY `donorVisitID` (`donorVisitID`),
  CONSTRAINT `sample_ibfk_1` FOREIGN KEY
(`donorVisitID`) REFERENCES `donor_visit`
(`donor_visit_id`)
)

CREATE TABLE `visit_update` (
  `updateID` varchar(50) NOT NULL,
  `visit` varchar(50) default NULL,
  PRIMARY KEY  (`updateID`),
  KEY `visit` (`visit`),
  CONSTRAINT `visit_update_ibfk_1` FOREIGN
KEY (`visit`) REFERENCES `donor_visit`
(`donor_visit_id`)
)
```

University Schemas-1 (UNIV-1)

Recipient Schema

```
CREATE TABLE `course` (
  `number` varchar(50) NOT NULL,
  `courseTitle` varchar(50) default NULL,
  `description` varchar(50) default NULL,
  `prerequisites` varchar(50) default NULL,
  `instructor` varchar(50) default NULL,
  PRIMARY KEY  (`number`),
  KEY `instructor` (`instructor`),
  CONSTRAINT `course_ibfk_1` FOREIGN KEY
(`instructor`) REFERENCES `faculty_member`
(`faculty_member_id`)
)

CREATE TABLE `faculty_member` (
  `faculty_member_id` varchar(50) NOT NULL,
  `personName` varchar(50) default NULL,
  `personTitle` varchar(50) default NULL,
  `homepage` varchar(50) default NULL,
  `researchInterest` varchar(50) default NULL,
  `email` varchar(50) default NULL,
  PRIMARY KEY  (`faculty_member_id`)
)

CREATE TABLE `paper` (
  `paperTitle` varchar(50) NOT NULL,
  `description` varchar(50) default NULL,
  `publicationYear` varchar(50) default NULL,
  PRIMARY KEY  (`paperTitle`),
  KEY `publicationYear` (`publicationYear`),
  CONSTRAINT `paper_ibfk_1` FOREIGN KEY
(`publicationYear`) REFERENCES `year` (`yr`)
)

CREATE TABLE `paper_author` (
  `paperTitle` varchar(50) NOT NULL,
  `author` varchar(50) NOT NULL,
  PRIMARY KEY  (`paperTitle`,`author`),
  KEY `FK_paper_author_3` (`author`),
  CONSTRAINT `FK_paper_author_3` FOREIGN
KEY (`author`) REFERENCES `student`
(`student_id`),
  CONSTRAINT `FK_paper_author_2` FOREIGN
KEY (`author`) REFERENCES `faculty_member`
(`faculty_member_id`),
  CONSTRAINT `paper_author_ibfk_1` FOREIGN
KEY (`paperTitle`) REFERENCES `paper`
(`paperTitle`)
)
```

Donor Schema

```
CREATE TABLE `academic_staff` (
  `academic_staff_id` varchar(50) NOT NULL,
  `name` varchar(50) default NULL,
  `office` varchar(50) default NULL,
  `email` varchar(50) default NULL,
  `phone` varchar(50) default NULL,
  PRIMARY KEY  (`academic_staff_id`)
)

CREATE TABLE `admin_staff` (
  `admin_staff_id` varchar(50) NOT NULL,
  `name` varchar(50) default NULL,
  `office` varchar(50) default NULL,
  `email` varchar(50) default NULL,
  `phone` varchar(50) default NULL,
  PRIMARY KEY  (`admin_staff_id`)
)

CREATE TABLE `areas_of_interest` (
  `interest_id` varchar(50) NOT NULL,
  `area` varchar(50) NOT NULL,
  PRIMARY KEY  (`interest_id`,`area`),
  CONSTRAINT `areasofinterest_ibfk_1` FOREIGN
KEY (`interest_id`) REFERENCES `student`
(`student_id`),
  CONSTRAINT `areasofinterest_ibfk_2` FOREIGN
KEY (`interest_id`) REFERENCES `academic_staff`
(`academic_staff_id`)
)

CREATE TABLE `course` (
  `courseNumber` varchar(40) NOT NULL,
  `courseTitle` varchar(50) default NULL,
  `instructor` varchar(50) default NULL,
  `area` varchar(50) default NULL,
  `description` varchar(200) default NULL,
  `prerequisite` varchar(200) default NULL,
  PRIMARY KEY  (`courseNumber`),
  KEY `instructor` (`instructor`),
  CONSTRAINT `course_ibfk_1` FOREIGN KEY
(`instructor`) REFERENCES `academic_staff`
(`academic_staff_id`)
)

CREATE TABLE `student` (
  `student_id` varchar(50) NOT NULL,
  `student_name` varchar(50) NOT NULL,
   `email` varchar(50) default NULL,
  `supervisor` varchar(50) default NULL,
```

```
CREATE TABLE `person_project` (
  `person` varchar(50) NOT NULL,
  `projectTitle` varchar(50) NOT NULL,
  PRIMARY KEY  (`person`,`projectTitle`),
  KEY `projectTitle` (`projectTitle`),
  CONSTRAINT `FK_person_project_3` FOREIGN
KEY (`person`) REFERENCES `student`
(`student_id`),
  CONSTRAINT `FK_person_project_2` FOREIGN
KEY (`person`) REFERENCES `faculty_member`
(`faculty_member_id`),
  CONSTRAINT `person_prject_ibfk_2` FOREIGN
KEY (`projectTitle`) REFERENCES `project`
(`projectTitle`)
)

CREATE TABLE `project` (
  `projectTitle` varchar(50) NOT NULL,
  `description` varchar(50) default NULL,
  `link` varchar(50) default NULL,
  PRIMARY KEY  (`projectTitle`)
)

CREATE TABLE `seminar` (
  `about` varchar(50) NOT NULL,
  `speaker` varchar(50) default NULL,
  `date` varchar(50) default NULL,
  `location` varchar(50) default NULL,
  PRIMARY KEY  (`about`),
  KEY `speaker` (`speaker`),
  CONSTRAINT `FK_seminar_1` FOREIGN KEY
(`speaker`) REFERENCES `faculty_member`
(`faculty_member_id`)
)

CREATE TABLE `student` (
  `student_id` varchar(50) NOT NULL,
  `studentName` varchar(50) default NULL,
  `advisor` varchar(50) default NULL,
  `email` varchar(50) default NULL,
  PRIMARY KEY  (`student_id`),
  KEY `advisor` (`advisor`),
  CONSTRAINT `FK_student_2` FOREIGN KEY
(`advisor`) REFERENCES `faculty_member`
(`faculty_member_id`)
)

CREATE TABLE `year` (
  `yr` varchar(50) NOT NULL,
  PRIMARY KEY  (`yr`)
)
```

```
  PRIMARY KEY  (`student_id`),
  KEY `supervisor` (`supervisor`),
  CONSTRAINT `student_ibfk_1` FOREIGN KEY
(`supervisor`) REFERENCES `academic_staff`
(`academic_staff_id`)
)
```

University Schemas-2 (UNIV-2)

Recipient Schema

```
CREATE TABLE `academic_programme` (
  `academic_programme_ID` int(11) NOT NULL,
  `ACADEMIC_YEAR` char(10) NOT NULL,
  `ACADEMIC_SEMESTER` varchar(50) NOT NULL,
  `PROGRAM_REF` int(11) default NULL,
  PRIMARY KEY  (`academic_programme_ID`),
  KEY `parent_programme` (`PROGRAM_REF`),
  CONSTRAINT `parent_programme` FOREIGN KEY
(`PROGRAM_REF`) REFERENCES `program`
(`program_ID`)
)

CREATE TABLE `academic_staff_member` (
  `academic_staff_member_ID` int(11) NOT NULL,
  `STAFF_NAME` varchar(150) NOT NULL,
  `STAFF_EMAIL` varchar(75) default NULL,
  `STAFF_PHONE` varchar(50) default NULL,
  `STAFF_FAX` varchar(75) default NULL,
  `STAFF_IDENTIFICATION_NUM` varchar(100) NOT
NULL,
  `STAFF_BIRTHDATE` date NOT NULL,
  PRIMARY KEY  (`academic_staff_member_ID`),
  UNIQUE KEY `ACADEMICSTAFF_EMAIL_UNIQUE`
(`STAFF_EMAIL`)
)

CREATE TABLE `campus` (
  `campus_ID` int(11) NOT NULL,
  `CAMPUS_NAME` varchar(150) NOT NULL,
  `CAMPUS_LOCATION` varchar(150) default NULL,
  `UNVCAMPUS` int(11) default NULL,
  PRIMARY KEY  (`campus_ID`),
  KEY `parentuniversity` (`UNIVERSITY_REF`),
  CONSTRAINT `parentuniversity` FOREIGN KEY
(`UNVCAMPUS`) REFERENCES `university`
(`university_ID`)
)

CREATE TABLE `course` (
  `course_ID` int(11) NOT NULL,
  `COURSE_NAME` varchar(150) NOT NULL,
  `COURSE_CREDITS` smallint(6) NOT NULL default '3',
  `COURSE_PROVIDER` int(11) NOT NULL,
  `COURSE_INSTRUCTOR` int(11) NOT NULL,
  PRIMARY KEY  (`course_ID`),
  KEY `parent_instructor` (`COURSE_INSTRUCTOR`),
  KEY `provider_department` (`COURSE_PROVIDER`),
  CONSTRAINT `parent_instructor` FOREIGN KEY
(`COURSE_INSTRUCTOR`) REFERENCES
`academic_staff_member` (`academic_staff_member_ID`),
  CONSTRAINT `provider_department` FOREIGN KEY
(`COURSE_PROVIDER`) REFERENCES `department`
(`department_ID`)
)
```

Donor Schema

```
CREATE TABLE `academic_course` (
  `academic_course_ID` int(11) NOT NULL,
  `ACADEMIC_COURSE_NAME`
varchar(150) NOT NULL,
  `ACADEMIC_COURSE_CREDITS`
smallint(6) NOT NULL default '3',
  `ACADEMIC_COURSE_PROVIDER`
int(11) NOT NULL,
  `ACADEMIC_COURSE_INSTRUCTOR`
int(11) NOT NULL,
  PRIMARY KEY  (`academic_course_ID`),
  KEY `parent_instructor`
(`ACADEMIC_COURSE_INSTRUCTOR`),
  KEY `provider_department`
(`ACADEMIC_COURSE_PROVIDER`),
  CONSTRAINT `parent_instructor`
FOREIGN KEY
(`ACADEMIC_COURSE_INSTRUCTOR`)
REFERENCES
`university_academic_instructor`
(`university_academic_instructor_ID`),
  CONSTRAINT `provider_department`
FOREIGN KEY
(`ACADEMIC_COURSE_PROVIDER`)
REFERENCES `department`
(`department_ID`)
)

CREATE TABLE `academic_institution` (
  `academic_institution_ID` int(11) NOT
NULL,
  `ACADEMIC_INSTITUTION_NAME`
varchar(150) NOT NULL,
  `ACADEMIC_INSTITUTION_WEBSITE`
varchar(150) NOT NULL,
  PRIMARY KEY
(`academic_institution_ID`)
)

CREATE TABLE `academic_programme` (
  `academic_programme_ID` int(11) NOT
NULL,
  `YEAR` char(10) NOT NULL,
  `SEMESTER` varchar(50) NOT NULL,
  `PROGRAM_REF` int(11) default NULL,
  PRIMARY KEY
(`academic_programme_ID`),
  KEY `parent_programme`
(`PROGRAM_REF`),
  CONSTRAINT `parent_programme`
FOREIGN KEY (`PROGRAM_REF`)
REFERENCES `program` (`program_ID`)
)

CREATE TABLE `department` (
```

```
CREATE TABLE `department` (
  `department_ID` int(11) NOT NULL,
  `DEPT_NAME` varchar(150) NOT NULL,
  `FACULTY_REF` int(11) NOT NULL,
  PRIMARY KEY  (`department_ID`),
  KEY `parent_faculty` (`FACULTY_REF`),
  CONSTRAINT `parent_faculty` FOREIGN KEY
(`FACULTY_REF`) REFERENCES `faculty` (`faculty_ID`)
)

CREATE TABLE `faculty` (
  `faculty_ID` int(11) NOT NULL,
  `FACULTY_NAME` varchar(150) NOT NULL,
  `DEAN_REF` int(11) NOT NULL,
  `UNIVERSITY_REF` int(11) NOT NULL,
  PRIMARY KEY  (`faculty_ID`),
  KEY `parent_dean` (`DEAN_REF`),
  KEY `parent_university` (`UNIVERSITY_REF`),
  CONSTRAINT `parent_dean` FOREIGN KEY
(`DEAN_REF`) REFERENCES `academic_staff_member`
(`academic_staff_member_ID`),
  CONSTRAINT `parent_university` FOREIGN KEY
(`UNIVERSITY_REF`) REFERENCES `university`
(`university_ID`)
)

CREATE TABLE `program` (
  `program_ID` int(11) NOT NULL,
  `PROGRAM_NAME` varchar(150) NOT NULL,
  `PROGRAM_DESC` varchar(150) default NULL,
  PRIMARY KEY  (`program_ID`)
)

CREATE TABLE `registration` (
  `registration_ID` int(11) NOT NULL,
  `REGISTRATION_ACADEMICSTAFFMEMBER_REF`
int(11) default NULL,
  `REGISTRATION_COURSE_REF` int(11) default NULL,
  `REGISTRATION_ACADEMICPROGRAMME_REF`
int(11) NOT NULL,
  PRIMARY KEY  (`registration_ID`),
  KEY `parent_academic_entity`
(`REGISTRATION_ACADEMICSTAFFMEMBER_REF`),
  KEY `parent_academic_programme`
(`REGISTRATION_ACADEMICPROGRAMME_REF`),
  KEY `parent_course`
(`REGISTRATION_COURSE_REF`),
  CONSTRAINT `parent_academic_entity` FOREIGN KEY
(`REGISTRATION_ACADEMICSTAFFMEMBER_REF`)
REFERENCES `academic_staff_member`
(`academic_staff_member_ID`),
  CONSTRAINT `parent_academic_programme` FOREIGN
KEY
(`REGISTRATION_ACADEMICPROGRAMME_REF`)
REFERENCES `academic_programme`
(`academic_programme_ID`),
  CONSTRAINT `parent_course` FOREIGN KEY
(`REGISTRATION_COURSE_REF`) REFERENCES
`course` (`course_ID`)
  `department_ID` int(11) NOT NULL,
  `DEPT_NAME` varchar(150) NOT NULL,
  `UNIVERSITY_REF` int(11) NOT NULL,
  PRIMARY KEY  (`department_ID`),
  KEY `parent_university`
(`UNIVERSITY_REF`),
  CONSTRAINT `parent_university`
FOREIGN KEY (`UNIVERSITY_REF`)
REFERENCES `academic_institution`
(`academic_institution_ID`)
)

CREATE TABLE `program` (
  `program_ID` int(11) NOT NULL,
  `PROGRAM_NAME` varchar(150) NOT
NULL,
  `PROGRAM_DESC` varchar(150) default
NULL,
  PRIMARY KEY  (`program_ID`)
)

CREATE TABLE
`university_academic_instructor` (
  `university_academic_instructor_ID` int(11)
NOT NULL,
  `NAME` varchar(150) NOT NULL,
  `ELECTRONIC_MAIL` varchar(75) default
NULL,
  `OFFICE_ADDRESS` varchar(150) default
NULL,
  `TELEPHONE` varchar(50) default NULL,
  PRIMARY KEY
(`university_academic_instructor_ID`)
)

CREATE TABLE `university_student` (
  `university_student_ID` int(11) NOT NULL,
  `NAME` varchar(150) NOT NULL,
  `ELECTRONIC_MAIL` varchar(75) default
NULL,
  `TELEPHONE` varchar(50) default NULL,
  PRIMARY KEY  (`university_student_ID`)
)
```

) CREATE TABLE `university` (`university_ID` int(11) NOT NULL, `UNIVERSITY_NAME` varchar(150) NOT NULL, `UNIVERSITY_WEBSITE` varchar(150) NOT NULL, `UNIVERSITY_ESTABLISHMENT_DATE` date default NULL, PRIMARY KEY (`university_ID`), UNIQUE KEY `NAME_CONSTRAINT` (`UNIVERSITY_NAME`))	

University Schemas-3 (UNIV-3)

Recipient Schema	**Donor Schema**
CREATE TABLE `address` (`Id` int(11) NOT NULL, `Street` text, `City` text, `PostalCode` int(11) default NULL, PRIMARY KEY (`Id`)) CREATE TABLE `payrate` (`Rank` int(11) NOT NULL, `HrRate` double default NULL, PRIMARY KEY (`Rank`)) CREATE TABLE `professor` (`Id` int(11) NOT NULL, `Name` text, `Sal` double default NULL, `addr` int(11) default NULL, PRIMARY KEY (`Id`), KEY `FK_professor_1` (`addr`), CONSTRAINT `FK_professor_1` FOREIGN KEY (`addr`) REFERENCES `address` (`Id`)) CREATE TABLE `student` (`Name` text, `GPA` double default NULL, `Yr` int(11) default NULL) CREATE TABLE `workson` (`Name` text, `Proj` text, `Hrs` int(11) default NULL, `ProjRank` int(11) default NULL, KEY `FK_workson_1` (`ProjRank`), CONSTRAINT `FK_workson_1` FOREIGN KEY (`ProjRank`) REFERENCES `payrate` (`Rank`))	CREATE TABLE `professor` (`Id` int(11) NOT NULL, `Name` text, `Salary` double default NULL, `Address` text, PRIMARY KEY (`Id`)) CREATE TABLE `student` (`Name` text, `GradePointAverage` double default NULL, `Year` int(11) default NULL) CREATE TABLE `workson` (`StudentName` text, `Project` text, `Expenses` double default NULL)

Appendix E

Evaluation of Schema Matching – For "select max above threshold" strategy

In the second type of experiments, we used the "select max above threshold" strategy. Results of precision, recall, f-measure, and overall measures for SASMINT and COMA++ are shown in Figures E.1 through E.8. In general, this strategy achieves better than the "select all above threshold" strategy, when precision, f-measure, and overall are considered. This is due to the fact that not all matches above the threshold are selected, but only those with higher similarity values. This brings about less number of false positives, which means precision is higher than the "select all above threshold" strategy. However in some cases, this strategy may lead to lower recall values because of missing some correct matches. This effect is little compared to the high increase in precision. Therefore, in general, the values for f-measure and overall were higher for both SASMINT and COMA++, when "select max above threshold" strategy was used, but again on average SASMINT performed slightly better then COMA++ .

E.1 Evaluation of Schema Matching Using Precision

Precision values for SASMINT and COMA++ were the same for the purchase order, UNIV-2, and UNIV-3 schemas. For the purchase order and UNIV-3 schemas, they both had the precision of 1.0. For hotel and UNIV-1 schemas, SASMINT performed around 1.08 times better than COMA++, but for the SDB schema, COMA++ had 1.05 times higher precision value. The reason for the difference between the performances of the two systems was because of the false positives introduced by the systems. For example, for the SDB schema, SASMINT identified "donorID" and "donorVisitID" as similar, which was incorrect and thus resulted in a decrease in the precision. On the average, SASMINT achieved 0.94 precision over all schema pairs, whereas the average precision of COMA++ was 0.93. Compared to the "select all above threshold" strategy, precision of "select max above threshold" strategy was very high for both systems. This is due to the fact that, in this second strategy only the most relevant matches were selected, which had the higher similarities than irrelevant matches. For example in the test with UNIV-2, although the first strategy identified "STAFF_EMAIL" column of the "academic_staff_member" table and "ELECTRONIC_MAIL" column of the "university_student" table as similar, the second strategy did not make this mistake. Figures E.1 and E.2 show the complete results for COMA++ and SASMINT.

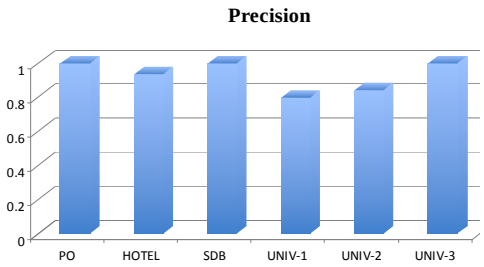


Fig. E.1. Precision values for **COMA++** - select max above threshold strategy

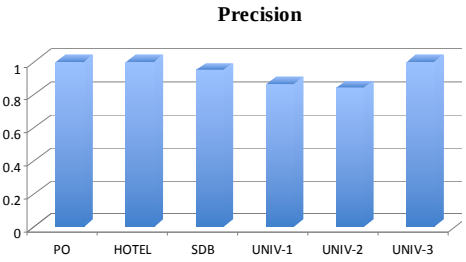


Fig. E.2. Precision values for **SASMINT** - select max above threshold strategy

E.2 Evaluation of Schema Matching Using Recall

Recall values in the case of "select max above threshold" strategy were either the same or a bit lower than the ones in the "select all above threshold" strategy for SASMINT. For COMA++, recall was lower in the "select max above threshold" strategy for all schema pairs. This was because of missing some correct matches. SASMINT and COMA++ both had the same recall values for the SDB and UNIV-2 schemas, as shown in Figures E.3. and E.4. For the hotel schemas, COMA++ was 1.1 times better than SASMINT. This was because of some table-to-table and column-to-column matches that could not be identified by SASMINT. Namely, similar to the case in "select all above threshold" strategy, these matching pairs were semantically similar, but since the current version of WordNet did not provide high "semantic" similarity values for these pairs, SASMINT could not identify them as similar. However, SASMINT achieved for the purchase order and UNIV-3 schemas 1.2 times and for the UNIV-1 schemas 1.1 times better than COMA++. On average, SASMINT had recall of 0.77, whereas COMA++ had 0.72.

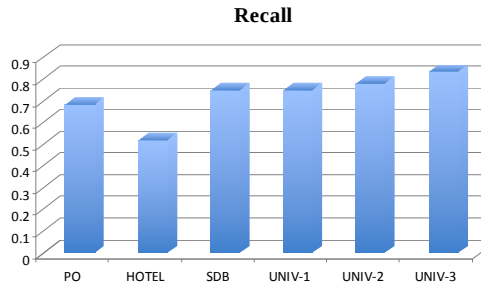


Fig. E.3. Recall values for **COMA++** - select max above threshold strategy

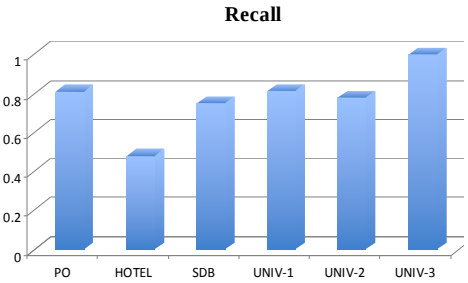


Fig. E.4. Recall values for **SASMINT** - select max above threshold strategy

E.3 Evaluation of Schema Matching Using F-Measure

When precision and recall values are combined using f-measure, SASMINT and COMA++ accomplished almost the same for the hotel, SDB, and UNIV-2 schemas. However, for the remaining schemas, SASMINT performed around 1.1 times better than COMA++. The

average f-measure for SASMINT was 0.84, whereas for COMA++ it was 0.80 and thus the quality of SASMINT's results is better than COMA++. F-measure values for COMA++ and SASMINT over all schema pairs are shown in Figures E.5 and E.6 respectively.

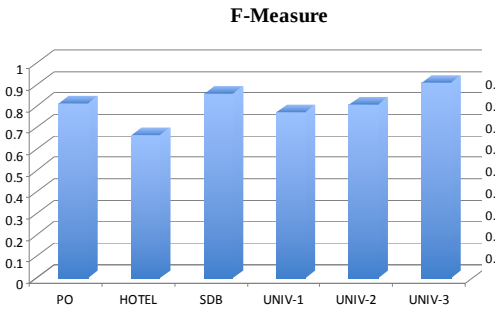


Fig. E.5. F-measure values for **COMA++** - select max above threshold strategy

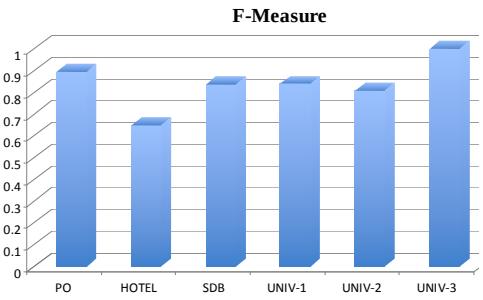


Fig. E.6. F-measure values for **SASMINT** - select max above threshold strategy

E.4 Evaluation of Schema Matching Using Overall

Situation for the overall measure was similar to f-measure, except for the SDB schema. For this pair, the value for COMA++ was slightly (1.05 times) better than SASMINT. For the hotel and UNIV-2 schemas, overall values for SASMINT and COMA++ were the same. However, for the purchase order, UNIV-1, and UNIV-3 schemas, SASMINT performed 1.2 times better than COMA++. The average overall value for SASMINT was 0.72, whereas for COMA++ it was 0.66. Complete results for COMA++ and SASMINT are shown in Figures E.7 and E.8 respectively.

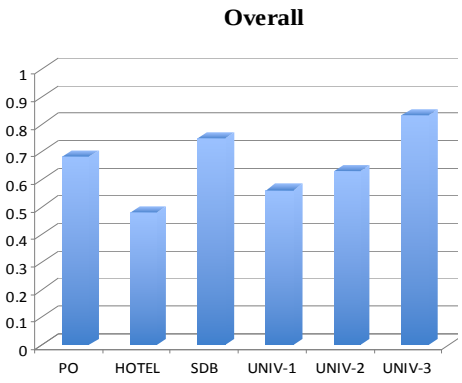


Fig. E.7. Overall values for **COMA++** - select max above threshold strategy

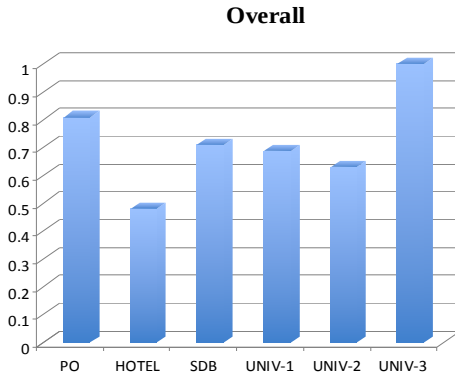


Fig. E.8. Overall values for **SASMINT** - select max above threshold strategy

Appendix F

Evaluation of Schema Integration - Details of Steps

As explained in Section 6.7, Schema Pair#4, #5 and #6 are used in the schema integration tests. In order to clarify the calculations for the completeness and minimality, we list below the number of concepts and keys in schemas of all these three schema pairs:

x1: number of concepts in the first schema of Schema Pair#4 = 39

a1: number of keys in the first schema of Schema Pair#4 = 21

y1: number of concepts in the second schema of Schema Pair#4 = 27

b1: number of keys in the second schema of Schema Pair#4 = 10

x2: number of concepts in the first schema of Schema Pair#5 = 47

a2: number of keys in the first schema of Schema Pair#5 = 19

y2: number of concepts in the second schema of Schema Pair#5 = 34

b2: number of keys in the second schema of Schema Pair#5 = 11

x3: number of concepts in the first schema of Schema Pair#6 = 22

a3: number of keys in the first schema of Schema Pair#6 = 5

y3: number of concepts in the second schema of Schema Pair#6 = 13

b3: number of keys in the second schema of Schema Pair#6 = 1

Step-1: First Schema of Schema Pair#5 + Second Schema of Schema Pair#5

We show in Table 6.2, the matches between two schemas of Pair#5. By exploiting these matches and using the integration rules explained in Chapter 4, SASMINT generated the first integrated schema, called Integrated Schema#1. The integrated schema in SASMINT is represented in the XML format, based on the SDML. SASMINT's XML representation of integrated schema specifies both the new elements of the integrated schema and how these

elements are derived from the elements of the two schemas being integrated. During the integration process, one redundancy was automatically generated, which was the "UNIVERSITY_REF" column of the "department" table. Therefore, the result of minimality measure was 0.99, as shown below, which is a substantial automated achievement.

$$m_{minimality} = 1 - \frac{n_{redundant}}{n_{total}} ==> 1 - \frac{1}{x2 + y2} ==> 1 - \frac{1}{47 + 34} \cong 0.99$$

When key minimality is considered, one redundant foreign key was generated on the same "UNIVERSITY_REF" column. Therefore, the key minimality is computed as 0.97, as shown below:

$$m_{minimalityKey} = 1 - \frac{n_{redundantKey}}{n_{totalKey}} ==> 1 - \frac{1}{a2 + b2} ==> 1 - \frac{1}{19 + 11} \cong 0.97$$

Although the resulting integrated schema had one redundant element and foreign key, it covered all the elements and keys of two source schemas. Therefore, the result was considered as 100% complete and 100% key complete, which is again a substantial automated achievement.

Step-2: Integrated Schema#1 + First Schema of Schema Pair#6

At the second step, using the matches that we identified between the Integrated Schema#1 and the first schema of the Schema Pair#6, SASMINT generated the Integrated Schema#2. There is no "OFFICE_ADDRESS" column in the Integrated Schema#2 anymore. This is due to the fact that the first schema of Schema Pair#6 had a table for the address information and the "professor" table had a foreign key to the "address" table. Since the "academic_staff_member" and "professor" are matched, in the new integrated schema, the "academic_staff_member" had a new foreign key to the "address" table, and therefore, "OFFICE_ADDRESS" column is replaced with a foreign key. Moreover, some new tables and columns were also added in the Integrated Schema#2. Since the "department" table still had the redundant "UNIVERSITY_REF" column and the foreign key, the results of minimality and key minimality were 0.99 and 0.97 respectively, as shown below:

$$m_{minimality} = 1 - \frac{n_{redundant}}{n_{total}} ==> 1 - \frac{1}{x2 + y2 + x3} ==> 1 - \frac{1}{47 + 34 + 22} \cong 0.99$$

$$m_{minimalityKey} = 1 - \frac{n_{redundantKey}}{n_{totalKey}} ==> 1 - \frac{1}{a2 + b2 + a3} ==> 1 - \frac{1}{19 + 11 + 5} \cong 0.97$$

Since all the concepts and keys of the three schemas (first and second schema of Schema Pair#5 and the first schema of Schema Pair#6) were represented in the integrated schema, completeness and key completeness were both 100% after this step.

Step-3: Integrated Schema#2 + Second Schema of Schema Pair#6

In this step, we first determined the matches between the Integrated Schema#2 and the second schema of the Schema Pair#6. SASMINT generated Integrated Schema#3, based on these matches. Considering the concepts (columns and tables) and keys, the resulting schema was

again complete. Redundant "UNIVERSITY_REF" column and the foreign key defined on it still existed after this step. Therefore, minimality and key minimality after this step were calculated as 0.99 and 0.97 respectively, as shown below:

$$m_{minimality} = 1 - \frac{n_{redundant}}{n_{total}} ==> 1 - \frac{1}{x2 + y2 + x3 + y3} ==> 1 - \frac{1}{47 + 34 + 22 + 13} \cong 0.99$$

$$m_{minimalityKey} = 1 - \frac{n_{redundantKey}}{n_{totalKey}} ==> 1 - \frac{1}{a2 + b2 + a3 + b3} ==> 1 - \frac{1}{19 + 11 + 5 + 1} \cong 0.97$$

Step-4: Integrated Schema#3 + First Schema of Schema Pair#4

In Step 4, we identified the matches among the elements of the Integrated Schema#3 and the first schema of the Schema Pair#4 and then integrated these two schema pairs. Resulting integrated schema is called Integrated Schema#4. Although a match was specified between the "Proj" column of the "workson" table of Integrated Schema#3 and the "project" table of the first schema of the Schema Pair#4, the resulting integrated schema missed a foreign key column in "workson" table referencing to the "project" table. Furthermore, the "author" column of the "paper-author" table originally had a foreign key reference to two different tables. However, in the resulting Integrated Schema#4, only one of them was kept. The same case happened to the "person" column of the "person-project" table. Considering the concepts schema was 100% complete, but since three foreign keys are missed, the key completeness decreased after this step, as shown in the calculations below. Redundancy was again due to the "UNIVERSITY_REF" column and the foreign key defined on it.

$$m_{minimality} = 1 - \frac{n_{redundant}}{n_{total}} ==> 1 - \frac{1}{x2 + y2 + x3 + y3 + x1} ==> 1 - \frac{1}{47 + 34 + 22 + 13 + 39} \cong 0.99$$

$$m_{minimalityKey} = 1 - \frac{n_{redundantKey}}{n_{totalKey}} ==> 1 - \frac{1}{a2 + b2 + a3 + b3 + a1} ==> 1 - \frac{1}{19 + 11 + 5 + 1 + 21} \cong 0.98$$

$$m_{completenessKey} = \frac{n_{completeKey}}{n_{totalKey}} ==> \frac{a2 + b2 + a3 + b3 + a1 - 3}{a2 + b2 + a3 + b3 + a1} ==> \frac{19 + 11 + 5 + 1 + 21 - 3}{19 + 11 + 5 + 1 + 21} \cong 0.95$$

Step-5: Integrated Schema#4 + Second Schema of Schema Pair#4

In the final step of schema integration, we identified the matches between Integrated Schema#4 and the second schema of the Schema Pair#4. There was a match between the "researchInterest" column of the "academic_staff_member" table and the "areas_of_interest" table of the second schema. Furthermore, in the original schema of Schema Pair#4, "interest_id" was a foreign key to the "student" and to the "academic_staff" tables. The "student" table matched the "university_student" table and the "academic_staff" table matched the "academic_staff_member" table of Integrated Schema#4. However, in the final integrated schema, these foreign key relationships were missed. The automated removal of "researchInterest" column was correct, but there had to be a foreign key reference from the "areas_of_interest" table to the "academic_staff_member" and "university_student" tables. Besides missing these two relationships, there was no concept of the recipient and donor

schemas that were not represented in the Integrated Schema#5, meaning that integration was 100% complete. Therefore, final integrated schema was 100% complete, 93% key complete, 99% minimal, and 99% key minimal as shown below. Redundancy was again due to the "UNIVERSITY_REF" column and the foreign key defined on it.

$$m_{minimality}=1-\frac{n_{redundant}}{n_{total}}==>1-\frac{1}{x2+y2+x3+y3+x1+yl}==>1-\frac{1}{47+34+22+13+39+27}\cong 0.99$$

$$m_{minimalityKey}=1-\frac{n_{redundantKey}}{n_{totalKey}}==>1-\frac{1}{a2+b2+a3+b3+a1+b1}==>$$
$$1-\frac{1}{19+11+5+1+21+10}\cong 0.99$$

$$m_{completenessKey}=\frac{n_{completeKey}}{n_{totalKey}}==>\frac{a2+b2+a3+b3+a1+b1-3-2}{a2+b2+a3+b3+a1+b1}==>$$
$$\frac{19+11+5+1+21-3-2}{19+11+5+1+21+10}\cong 0.93$$

Bibliography

Afsarmanesh, H., Benabdelkader, A., & Hertzberger, L. O. (1998). A flexible approach to information sharing in water industries. In: *Proceedings of the International Conference on Information Technology - CIT98*, Bhubaneswar, India. Tata McGraw-Hill Publishing Company Limited, pp.135-142.

Afsarmanesh, H., & Camarinha-Matos, L. M. (1997). Federated information management for cooperative virtual organizations. In: *Proceedings of the VIII International Conference on Database and Expert System Applications - DEXA'97*, Toulouse, France. Springer Verlag Lecture Notes in Computer Science (LNCS 1308), pp.561-572.

Afsarmanesh, H., & Camarinha-Matos, L. M. (2005). A framework for management of virtual organizations breeding environments. In: *Proceedings of PRO-VE'05-Collaborative Networks and their Breeding Environment, Valencia, Spain*. pp.35-48.

Afsarmanesh, H., Camarinha-Matos, L. M., & Ermilova, E. (2008). Vbe reference framework. In L. M. Camarinha-Matos, H. Afsarmanesh & M. Ollus (Eds.), *Methods and tools for collaborative networked organizations.* (pp. 35-68). 978-0-387-79423-5, New York, Springer.

Afsarmanesh, H., Guevara-Masis, V., & Hertzberger, L. O. (2004). Federated management of information for telecare. In: *Proceedings of the TELECARE 2004 - Int. Workshop on Tele-Care and Collaborative Virtual Communities in Elderly Care*, Porto, Portugal. INSTICC Press, pp.49-62, ISBN:972-8865-10-4.

Afsarmanesh, H., Wiedijk, M., Hertzberger, L. O., Gomes, F. J. N., Provedel, A., Martins, R. C., et al. (1996). Cooperation of cim expert systems supported by peer. *Special issue of Journal of Studies in Informatics and Control, 5*(2), pp.157-169.

Afsarmanesh, H., Wiedijk, M., Tuijnman, F., Bergman, M., & Trenning., P. (1994). *The peer information management language user manual*, Technical Report, CS-94-14, Dept. of Computer Science, University of Amsterdam.

An, Y., Mylopoulos, J., & Borgida, A. (2006). Building semantic mappings from databases to ontologies. In: *Proceedings of the Twenty-First National Conference on Artificial Intelligence - (AAAI-06)*, Boston, Massachusetts, USA. AAAI Press, pp.1557-1560, ISBN:978-1-57735-281-5.

Arens, Y., Knoblock, C. A., & Shen, W.-M. (1996). Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems, 6*(2/3), pp.99-130.

Aumueller, D., Do, H. H., Massmann, S., & Rahm, E. (2005). Schema and ontology matching with coma++. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Baltimore, Maryland, USA. ACM, pp.906-908, ISBN:1-59593-060-4.

Banerjee, S., & Pedersen, T. (2002). An adapted lesk algorithm for word sense disambiguation using wordnet. In: *Proceedings of the Third International Conference on Intelligent Text Processing and Computational Linguistics*, Mexico City - Mexico. Springer, pp.136-145.

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

Bayardo, R. J., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., et al. (1997). Infosleuth: Agent-based semantic integration of information in open and dynamic environments. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, USA. ACM, pp.195-206.

Beneventano, D., & Bergamaschi, S. (2004). The momis methodology for integrating heterogeneous data sources. In: *Proceedings of the IFIP Congress Topical Sessions*, Toulouse, France. Kluwer, pp.19-24, ISBN:1-4020-8156-1.

Bergamaschi, S., Castano, S., Beneventano, D., & Vincini, M. (2001). Semantic integration of heterogeneous information sources. *Data and Knowledge Engineering, 36*(1), pp.215-249.

Bergamaschi, S., Castano, S., Vimercati, S. D. C. d., Montanari, S., & Vincini, M. (1998). A semantic approach to information integration: The momis project. In: *Proceedings of the Sesto Convegno della Associazione Italiana per l'Intelligenza Artificiale - AI*IA98*, Padova, Italy.

Bernstein, P. A., Melnik, S., Petropoulos, M., & Quix, C. (2004). Industrial-strength schema matching. *SIGMOD Record, 33*(4), pp.38-43.

BizTalk. (2010). *Microsoft corporation: Biztalk mapper*, Last accessed 2010, from http://www.microsoft.com/biztalk.

Blondel, V. D., Gajardo, A., Heymans, M., Senellart, P., & Dooren, P. V. (2004). A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM Review, 46*(4), pp.647-666.

Brachman, R. J., & Schmolze, J. G. (1985). An overview of the kl-one knowledge representation system. *Cognitive Science, 9*(2), pp.171-216.

Brodie, M. L., & Ceri, S. (1992). On intelligent and cooperative information systems: A workshop summary. *International Journal of Intelligent and Cooperative Information Systems, 2*(2), pp.249-290.

Budanitsky, A., & Hirst, G. (2001). Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. In: *Proceedings of the Workshop on WordNet and Other Lexical Resources, Second meeting of the North American Chapter of the Association for Computational Linguistics*, Pittsburgh, USA. pp.29-34.

Bukhres, O., & Elmagarmid, A. (Eds.). (1996). *Object-oriented multidatabase systems: A solution for advanced applications*. Englewood Cliffs, NJ: Prentice Hall.

Bunke, H. (2000). Graph matching: Theoretical foundations, algorithms and applications. In: *Proceedings of the International Conference on Vision Interface*, Montreal, Quebec, Canada. pp.82-88.

Busse, S., Kutsche, R.-D., Leser, U., & Weber, H. (1999). *Federated information systems: Concepts, terminology and architectures,*Technical report, 99-9, Technische Universitat Berlin.

Camarinha-Matos, L. M., & Afsarmanesh, H. (1999a). The prodnet goals and approach. In: *Proceedings of PRO-VE'99- Infrastructures for Virtual Enterprises*, Porto, Portugal. Kluwer Academic Publishers, pp.97-108, ISBN: 0-7923-8639-6.

Camarinha-Matos, L. M., & Afsarmanesh, H. (1999b). The virtual enterprise concept. In: *Proceedings of the Infrastructures for Virtual Enterprises*, Porto, Portugal. Kluwer Academic Publishers, pp.3-14, ISBN:0-7923-8639-6.

Camarinha-Matos, L. M., & Afsarmanesh, H. (2003). Designing the information technology subsystem for enterprise integration. In P. Bernus, L. Nemes & G. Schmidt (Eds.), *Invited chapter in handbook on enterprise architecture* (pp. 617-680), ISBN: 3-540-00343-6, Springer.

Camarinha-Matos, L. M., & Afsarmanesh, H. (2008a). Classes of collaborative networks. In G. Putnik & M. M. Cunha (Eds.), *Encyclopedia of networked and virtual organizations*. Idea Group.

Camarinha-Matos, L. M., & Afsarmanesh, H. (2008b). Concept of collaboration. In G. Putnik & M. M. Cunha (Eds.), *Encyclopedia of networked and virtual organizations*. Idea Group.

Camarinha-Matos, L. M., Afsarmanesh, H., & Erbe, H. (Eds.) (2000). *Advances in networked enterprises - virtual organizations, balanced automation, and systems integration*. ISBN: 0-7923-7958-6, Boston, Kluwer Academic Publishers.

Camarinha-Matos, L. M., Afsarmanesh, H., & Ollus, M. (2005). Ecolead: A holistic approach to creation and management of dynamic virtual organizations. In: *Proceedings of PRO-VE'05- Collaborative Networks and their Breeding Environments*, Valencia, Spain. Springer, pp.3-16.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., & Rice, J. P. (1998). Okbc: A programmatic foundation for knowledge base interoperability. In: *Proceedings of the Artificial intelligence/Innovative applications of artificial intelligence*, Madison, Wisconsin, USA. American Association for Artificial Intelligence, pp.600-607.

Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., et al. (1994). The tsimmis project: Integration of heterogeneous information sources. In: *Proceedings of the 10th Meeting of the Information Processing Society of Japan, Tokyo, Japan*.pp.7-18.

Chiticariu, L., Hernández, M. A., Kolaitis, P. G., & Popa, L. (2007). Semi-automatic schema integration in clio. In: *Proceedings of the International Conference on Very Large Data Bases (VLDB) (Demo Track)*, Vienna, Austria. pp.1326-1329.

Chiticariu, L., Kolaitis, P. G., & Popa, L. (2008). Interactive generation of integrated schemas. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Vancouver, BC, Canada. ACM, pp.833-846.

Cleverdon, C. W., & Keen, E. M. (1966). *Factors determining the performance of indexing systems*. Cranfield, England, Aslib-Cranfield research project.

Cohen, W., Ravikumar, P., & Fienberg, S. E. (2003). A Comparison of String Distance Metrics for Name-Matching Tasks. In: *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web, Acapulco, Mexico*. pp.73-78.

Dhamankar, R., Lee, Y., Doan, A., Halevy, A., & Domingos, P. (2004). Imap: Discovering complex semantic matches between database schemas. In: *Proceedings of the ACM Sigmod Conference on Management of Data*, Paris, France. ACM, pp.383-394, ISBN:1-58113-859-8.

Do, H. H., Melnik, S., & Rahm, E. (2002). Comparison of schema matching evaluations. In: *Proceedings of the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*. Springer Lecture Notes in Computer Science (LNCS 2593), pp.221-237, ISBN:3-540-00745-8.

Do, H. H., & Rahm, E. (2002). Coma - a system for flexible combination of schema matching approaches. In: *Proceedings of the International Conference on Very Large Databases (VLDB)*, Hong Kong, China. pp.610-621.

Doan, A. H., Domingos, P., & Halevy, A. (2001). Reconciling schemas of disparate data sources - a machine-learning approach. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Santa Barbara, California, USA. ACM, pp.509-520.

Doan, A. H., Madhavan, J., Domingos, P., & Halevy, A. (2002). Learning to map between ontologies on the semantic web. In: *Proceedings of the World-Wide Web Conference*, Honolulu, Hawaii, USA. pp.662-673.

Dou, D., McDermott, D., & Qi, P. (2003). Ontology translation on the semantic web. In: *Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics - ODBASE'03*. Springer Lecture Notes in Computer Science (LNCS 2888), pp.952-969.

Duchateau, F., Bellahsene, Z., & Hunt, E. (2007). Xbenchmatch: A benchmark for xml schema matching tools. In: *Proceedings of the International Conference on Very Large Databases (VLDB)*, Vienna, Austria. pp.1318-1321, ISBN:978-1-59593-649-3.

Eclipse. (2010). *Eclipse*, Last accessed 2010, from http://www.eclipse.org/.

Ehrig, M., & Staab, S. (2004). Qom - quick ontology mapping. In: *Proceedings of the International Semantic Web Conference (ISWC)*, Hiroshima, Japan. Springer Lecture Notes in Computer Science (LNCS 3298), pp.683-697.

Ehrig, M., & Sure, Y. (2004). Ontology mapping - an integrated approach. In: *Proceedings of the European Semantic Web Symposium (ESWS)*, Heraklion, Crete, Greece. Springer Lecture Notes in Computer Science (LNCS 3053), pp.76-91, ISBN:3-540-21999-4.

Ehrig, M., & Sure, Y. (2005). Foam - framework for ontology alignment and mapping - results of the ontology alignment evaluation initiative. In: *Proceedings of the K-CAP (International Conference on Knowledge Capture) 2005 Workshop on Integrating Ontologies*, Banff, Canada. pp.72-76.

Elmagarmid, A., & Pu, C. (1990). Guest editors' introduction to the special issue on heterogeneous databases. *ACM Computing Surveys, 22*(3), pp.175-178.

Embley, D. W., Xu, L., & Ding, Y. (2004). Automatic direct and indirect schema mapping: Experiences and lessons learned. *SIGMOD Record, 33*(4), pp.14-19.

Fellbaum, C. (1998). *An electronic lexical database.*, Cambridge, MA, MIT press.

Foggia, P., Sansone, C., & Vento, M. (2001). A performance comparison of five algorithms for graph isomorphism. In: *Proceedings of the IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition*, Ischia, Italy. pp.188-199.

Gal, A. (2006). Managing uncertainty in schema matching with top-k schema mappings. *Journal on Data Semantics VI: Special Issue on Emergent Semantics, v. 4090*, pp.90-114.

Gal, A. (2007). Why is schema matching tough and what can we do about it? *SIGMOD Record, 35*(4), pp.2-5.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., et al. (1997). The tsimmis approach to mediation: Data models and languages. *Journal of Intelligent Information Systems, 8*(2), pp.117-132.

Giunchiglia, F., Shvaiko, P., & Yatskevich, M. (2004). S-match: An algorithm and an implementation of semantic matching. In: *Proceedings of the European Semantic Web Symposium (ESWS)*, Heraklion, Crete, Greece. Springer Lecture Notes in Computer Science, pp.61-75, ISBN:3-540-21999-4.

Goh, C., Bresson, S., Madnich, S., & Siegel, M. (1999). Context interchange: New features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems, 17*(3), pp.270-293.

GraphML. (2010). *Graphml*, Last accessed 2010, from http://graphml.graphdrawing.org/.

Gregor, R. M. M. (1988). A deductive pattern matcher. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Saint Paul, Minnesota, USA. Morgan Kaufmann Publisher, pp.403-408.

Gruber, T. R. (1993). A translation approach to portable ontologies. *Knowledge Acquisition, 5*(2), pp.199-220.

Guevara-Masis, V., Unal, O., Kaletas, E. C., Afsarmanesh, H., & Hertzberger, L. O. (2004). Using ontologies for collaborative information management: Some challenges & ideas. In: *Proceedings of the Third Biennial International Conference on Advances in Information Systems (ADVIS)*, Izmir, Turkey. Springer Lecture Notes in Computer Science (LNCS 3261), pp.107-116, ISBN:3-540-23478-0.

GXL. (2010). *Graph exchange language (gxl)*, Last accessed 2010, from http://www.gupro.de/GXL/.

Hammer, J., & McLeod, D. (1993). An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous database systems. *International Journal of Intelligent & Cooperative Information Systems, World Scientific, 2*(1), pp.51-83.

Hammer, M., & McLeod, D. (1979). *On database management system architecture*, Technical Report MIT/LCS/TM-141, MIT Lab for Computer Science.

Hammer, M., & Mcleod, D. (1981). Database description with sdm: A semantic database model. *ACM Transactions on Database Systems, 6*(3), pp.351-386.

Heimbigner, D., & McLeod, D. (1985). A federated architecture for information management. *ACM Transaction on Information Systems, 3*(3), pp.253-278.

Hirst, G., & St-Onge, D. (1998). Lexical chains as representations of context for the detection and correction of malapropisms. In C. Fellbaum (Ed.), *Wordnet: An electronic lexical database and some of its applications.*, Cambridge, MIT Press.

Huhns, M., Jacobs, N., Ksiezyk, T., Shen, W., Singh, M., & Cannata, P. (1992). Enterprise information modeling and model integration in carnot. In C. J. Petrie (Ed.), *Enterprise integration modeling* (pp. 290-299). MIT Press.

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

Jaro, M. A. (1995). Probabilistic linkage of large public health data files. *Statistics in Medicine, v. 14*, pp.491-498.

JGraph. (2010). *Jgraph*, Last accessed 2010, from http://www.jgraph.com/.

JGraphT. (2010). *Jgrapht*, Last accessed 2010, from http://jgrapht.sourceforge.net/.

Jiang, J. J., & Conrath, D. W. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. In: *Proceedings of the International Conference on Research in Computational Linguistics.*, Taipei, Taiwan. pp.19-33.

JWNL. (2010). *Jwnl*, Last accessed 2010, from http://jwordnet.sourceforge.net/.

Kahng, J., & McLeod, D. (2001). Dynamic classificational ontologies. In M. A. Arbib & G. J.S. (Eds.), *Computing the brain: A guide to neuroinformatics* (pp. 241-254). Academic Press.

Kamel, M. N., & Kamel, N. (1992). Federated database management system: Requirements, issues and solutions. *Computer Communications, 15*(4), pp.270-280.

Kirkpatrick, B. (1998). *Roget's thesaurus of english words and phrases.*, Harmondsworth, Middlesex, England, Penguin.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM, 46*(5), pp.604-632.

Leacock, C., & Chodorow., M. (1998). Combining local context with wordnet similarity for word sense identification. In C. Fellbaum (Ed.), *Wordnet: A lexical reference system and its application.* Cambridge, MIT Press.

Lenzerini, M. (2002). Data integration: A theoretical perspective. In: *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems - PODS*, Madison, Wisconsin. ACM, pp.233-246, ISBN:1-58113-507-6.

Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries: How to tell a pine code from an ice cream cone. In: *Proceedings of the 5th International Conference on Systems Documentation*, Toronto, Ontario, Canada. pp.24-26.

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

Levy, A. Y., Rajaraman, A., & Ordille, J. J. (1996). Querying heterogeneous information sources using source descriptions. In: *Proceedings of the International Conference on Very Large Database (VLDB)*, Bombay, India. Morgan Kaufmann, pp.251-262, ISBN:1-55860-382-4.

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

Litwin, W., & Abdellatif, A. (1986). Multidatabase interoperability. *Computer, 19*(12), pp.10-18.

Madhavan, J., Bernstein, P. A., & Rahm, E. (2001). Generic schema matching with cupid. In: *Proceedings of the International Conference on Very Large Databases (VLDB)*, Roma, Italy. Morgan Kaufmann, pp.49-58, ISBN:1-55860-804-4.

McGuinness, D. L., Fikes, R., Rice, J., & Wilder, S. (2000). An environment for merging and testing large ontologies. In: *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, Breckenridge, Colorado. pp.483-493.

Melnik, S., Garcia-Molina, H., & Rahm, E. (2002). Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: *Proceedings of the International Conference on Data Engineering (ICDE)*, San Jose, CA, USA. IEEE Computer Society, pp.117-128.

Melnik, S., Rahm, E., & Bernstein, P. A. (2003). Rondo: A programming platform for generic model management. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Diego, California, USA. ACM, pp.193-204, ISBN:1-58113-634-X.

Mena, E., Illarramendi, A., Kashyap, V., & Sheth, A. (2000). Observer: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. *Distributed and Parallel Databases Journal, 8(2)*, pp.223-271.

Miller, R. J., Haas, L. M., & Hernandez, M. A. (2000). Schema mapping as query discovery. In: *Proceedings of the International Conference on Very Large Databases (VLDB)*, Cairo, Egypt. Morgan Kaufmann, pp.77-88, ISBN:1-55860-715-3.

Mitra, P., Wiederhold, G., & Decker, S. (2001). A scalable framework for the interoperation of information sources. In: *Proceedings of the International Semantic Web Working Symposium (SWWS)*, California, USA. IOS press, pp.317-329, ISBN:1-58603-255-0.

Monge, A. E., & Elkan, C. (1996). The field matching problem: Algorithms and applications. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, USA. AAAI Press, pp.267-270, ISBN:1-57735-004-9.

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

Noy, N. F., & Musen, M. A. (2000). Prompt: Algorithm and tool for automated ontology merging and alignment. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, Texas, USA. AAAI Press/The MIT Press, pp.450-455, ISBN:0-262-51112-6.

Noy, N. F., & Musen, M. A. (2003). The prompt suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies, 59*(6), pp.983-1024.

Ozsu, T., & Valduriez, P. (1999). *Principles of distributed database systems*. ISBN: 0-13-659707-6, New Jersey, Prentice Hall.

Papakonstantinou, Y., Garcia-Molina, H., & Ullman, J. (1996). Medmaker: A mediation system based on declarative specifications. In: *Proceedings of the International Conference on Data Engineering (ICDE)*, New Orleans, Louisiana, USA. IEEE Computer Society, pp.132-141, ISBN:0-8186-7240-4.

Papakonstantinou, Y., Garcia-Molina, H., & Widom, J. (1995). Object exchange across heterogeneous information sources. In: *Proceedings of the International Conference on Data Engineering (ICDE)*, Taipei, Taiwan. IEEE Computer Society, pp.251-260, ISBN:0-8186-6910-1.

Patwardhan, S. (2003). *Incorporating dictionary and corpus information into a context vector measure of semantic relatedness.*Master's thesis, University of Minnesota.

Pedersen, T., Banerjee, S., & Patwardhan, S. (2005). *Maximizing semantic relatedness to perform word sense disambiguation.*, Research Report UMSI 2005/25, University of Minnesota Supercomputing Institute.

Pottinger, R., & Bernstein, P. A. (2008). Schema merging and mapping creation for relational sources. In: *Proceedings of the International Conference on Extending Database Technology (EDBT)*, Nantes, France. ACM, pp.73-84, ISBN:978-1-59593-926-5.

Pottinger, R. A., & Bernstein, P. A. (2003). Merging models based on given correspondences. In: *Proceedings of the International Conference on Very Large Databases (VLDB)*, Berlin, Germany. Morgan Kaufmann, pp.826-873, ISBN:0-12-722442-4.

Protege. (2010). *Protege*, Last accessed 2010, from http://protege.stanford.edu/.

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

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

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*. Montréal, Québec, Canada. pp.448-453.

Resnik, P. (1999). Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research (JAIR), 11*, pp.95-130.

Rijsbergen, C. J. V. (1979). *Information retrieval*. London, U.K., Butterworth.

Saleem, K., Bellahsene, Z., & Hunt, E. (2008). Porsche: Performance oriented schema mediation. *Information Systems, 33*(7-8), pp.637-657.

Salton, G., & Yang, C. S. (1973). On the specification of term values in automatic indexing. *Journal of Documentation, 29*, pp.351-372.

SecondString. (2010). *Secondstring*, Last accessed 2010, from http://secondstring.sourceforge.net/.

Sheth, A. (1998). Changing focus on interoperability in information systems: From system, syntax, structure to semantics. In M. F. Goodchild, M. J. Egenhofer, R. Fegeas & C. A. Kottman (Eds.), *Interoperating geographic information systems*. Kluwer.

Sheth, A., & Kashyap, V. (1992). So far (schematically), yet so near (semantically). In: *Proceedings of the Proceedings of the IFIP WG2.6 Conference on Semantics of Interoperable Database Systems*, Lorne, Victoria, Australia. North-Holland Publishing, pp.283-312, ISBN:0-444-89879-4.

Sheth, A., & Larson, J. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys, 22*(3), pp.183-236.

Shvaiko, P., & Euzenat, J. (2005). A survey of schema-based matching approaches. *Journal of Data Semantics IV, v. 3730*, pp.146-171.

Silberschatz, A., Stonebraker, M., & Ullman, J. D. (1990). Database systems: Achievements and opportunities. *SIGMOD Record, 19*(4).

Sørensen, C. (2005). *This is not an article-just some thoughts on how to write one*, Technical Report 121, London School of Economics and Political Science. United Kingdom.

Spaccapietra, S., Parent, C., & Dupont, Y. (1992). Model independent assertions for integration of heterogeneous schemas. *VLDB Journal, 1*(1), pp.81-126.

Tsichritzis, D. (1981). Integrating data base and message systems. In: *Proceedings of the International Conference on Very large Data Bases (VLDB)*, Cannes, France. IEEE Computer Society, pp.356-362.

Tuijnman, F., & Afsarmanesh, H. (1993). Management of shared data in federated cooperative peer environment. *International Journal of Intelligent and Cooperative Information Systems (IJICIS), 2*(4), pp.451-473.

Unal, O., & Afsarmanesh, H. (2006a). Interoperability in collaborative network of biodiversity organizations. In: *Proceedings of PRO-VE - Network-Centric Collaboration and Supporting Frameworks*, Helsinki, Finland. Springer, pp.515-524.

Unal, O., & Afsarmanesh, H. (2006b). Sasmint system for database interoperability in collaborative networks. Springer Lecture Notes in Computer Science (LNCS 4275), Springer, pp.91-108, ISBN:978-3-540-48287-3.

Unal, O., & Afsarmanesh, H. (2006c). Using linguistic techniques for schema matching. In: *Proceedings of the International Conference on Software and Data Technologies (ICSOFT)*, Setubal, Portugal. INSTICC Press, pp.115-120, ISBN:972-8865-69-4.

Unal, O., & Afsarmanesh, H. (2009). Schema matching and integration for data sharing among collaborating organizations. *Journal of Software, 4*(3), ISSN:1796-217X, pp.248-261.

Unal, O., & Afsarmanesh, H. (2010). Semi-automated schema integration with sasmint. *Journal of Knowledge and Information Systems., 23*(1), ISSN:0219-1377, pp.99-128.

Unal, O., Kaletas, E. C., Afsarmanesh, H., Yakali, H. H., & Hertzberger, L. O. (2005). Collaborative information management system for science domains. In S. Dasgupta (Ed.), *Encyclopedia of virtual communities and technologies*. Idea Group Publishing.

Wang, G., Goguen, J., Nam, Y., & Lin, K. (2004). Critical points for interactive schema matching. In: *Proceedings of the Sixth Asia Pacific Web Conference*, Hangzhou, China. Springer Lecture Notes in Computer Science, pp.654-664, ISBN:3-540-21371-6.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer, 25*(3), pp.38-49.

WordNet. (2010). *Wordnet*, Last accessed 2010, from http://www.cogsci.princeton.edu/~wn/.

Wu, Z., & Palmer, M. (1994). Verb semantics and lexical selection. In: *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, Las Cruces, New Mexico. Association for Computational Linguistics, pp.133-138.

XMLBeans. (2010). *Xmlbeans*, Last accessed 2010, from http://xmlbeans.apache.org/.

Zager, L. (2005). *Graph similarity and matching* S.M. Thesis, Massachusetts Institute of Technology.

Zisman, A. (1995). *Towards interoperability in heterogeneous database systems*, Technical Report 11, Department of Computing, Imperial College of Science, Technology and Medicine.

Summary

On semi-automated matching and integration of database schemas

Today, increasingly more organizations understand the need to collaborate in order to better achieve their common goals. As a result of this tendency towards increased collaboration, a line of research and development has focused on addressing data sharing and interoperation among organizations, and developing ICT tools and systems to support them. But many open challenges still remain in this area. Focusing on sharing and integration of information among independent nodes within collaborative networks, and to provision transparent access to the information stored in their databases, form the base for their effective co-working. But clearly, independent nodes model their information heterogeneously in their database schemas. Thus, before any information sharing can occur among these databases, the main challenges to be addressed include: (i) *identification and establishment of correspondences among concepts* defined in independent database schemas, (ii) *resolution of existing heterogeneities among database schemas* of the involved nodes, and (iii) *integration of these database schemas.* Typically these three tasks are complicated, even to handle manually, and this difficulty intensifies by the number of nodes within the collaborative network, as well as the size of their database schemas. A number of research approaches and prototypes have therefore aimed at automating the matching of schemas, while a few others have independently attempted at automating the integration of schemas. However, in spite of the related past efforts both in research and in commercial developments, today the matching and integration of schemas still involve a large amount of manual work and there are a large number of open research issues that remain in these areas.

This thesis proposes an *automated but supervised combined approach that addresses and merges the problems of matching and integration of relational database schemas*. Thus the main contribution of the thesis is a new combined schema matching/integration approach. A proof of concept for this approach is provided, as an implemented prototype system called SASMINT – Semi-Automatic Schema Matching and INTegration. The SASMINT system automatically identifies a large number of syntactic, semantic, and structural heterogeneities among relational database schemas. It then attempts to resolve their heterogeneity, proposing for user validation a list of potential matches among the compared schemas. The system then automatically generates an integrated schema from this list.

The Chapter 1 of this thesis presents the motivation for this work, the main research questions, the objectives, and the main contributions of this research.

Chapter 2 elaborates on the base definitions and the classification of concepts from the state of the art, in relation to architectures, approaches, and systems that enable sharing and exchange of distributed and heterogeneous information.

As heterogeneity stands at the center of the schema matching and integration processes, identification and resolution of different types of heterogeneities are crucial for the success of these processes. Chapter 3 presents different taxonomies introduced for heterogeneities, and narrows them down to the database schema heterogeneity, the main subject of the thesis.

After establishing the base for our research in the first three chapters, the rest of this dissertation addresses the design and development of our proposed approach. As a main chapter of this thesis, Chapter 4 contains a literature survey focused on (i) database integration and interoperability, (ii) database schema matching, (iii) database schema integration, and (iv) ontology matching and merging. Then our proposed solution, SASMINT, is introduced, to address a number of identified open issues. The SASMINT approach increases the accuracy of schema matching, through the weighted combination of a number of schema matching algorithms, where each algorithm resolves a different specific kind of syntactic, semantic, or structural conflicts. Furthermore, another contribution of SASMINT covered in Chapter 4 is the introduction of our so called SAMPLER technique, which semi-automatically identifies for every target domain the appropriate weights for each algorithm planned to be applied in the linguistic matching process. Chapter 4 also introduces an overview of a set of rules that enable the automatic generation of both the integrated schemas as well as the derivation constructs that represent the history of the integration process in the collaboration network. Our development of SASMINT Derivation Markup Language (SDML), which captures and supports the creation of both persisting schema match results and persisting schema integration results, is also described in this chapter**.**

Similar to any other research work, it is important to verify and validate the approach proposed by our research. For this purpose, we have implemented our approach in the SASMINT system, which is the focus of Chapter 5. The main components of the SASMINT system, as well as its operation are described in this chapter. While in our opinion, supporting user interactions through a GUI is fundamental for the semi-automated schema matching and schema integration processes, this component is typically missing from the research and development work in this area. A GUI is required to support the user with verification of the automatically identified schema conflicts and matches, as well as the proposed integrated schema. Through SASMINT's GUI, users interact with the system, set proper weights for its processes, approve/modify/disapprove its automatically generated results, and save the results of both schema matching and schema integration.

To demonstrate and evaluate the results of this research and to measure the quality of the SASMINT system, we have carried out a number of experiments. Specifically, the schema matching component of SASMINT is compared and evaluated against another state of the art schema matching system. But the approach of SASMINT is unique in that it merges the schema matching and schema integration processes. Hence, we could not find a counterpart to compare the schema integration component of SASMINT. Therefore, in our evaluation experiments we have only measured its success rate in producing accurate results. These experiments are elaborated at length in Chapter 6 of this thesis.

This thesis concludes by explaining how it has addressed the main research questions.

Samenvatting

Semiautomatische Vergelijking en Integratie van Database Schema's[1]

Tegenwoordig zien steeds meer organisaties het belang in van samenwerking om hun gezamenlijke doelstellingen beter te kunnen realiseren. Als een gevolg van deze sterker wordende impuls tot samenwerken zijn er onderzoeks- en ontwikkelingsgebieden ontstaan die zich richten op de studie naar het delen van gegevens en de coöperatie tussen organisaties, en het ontwikkelen van ICT gereedschap en systemen ter ondersteuning hiervan. Er bestaan echter nog vele open uitdagingen op dit gebied. Een focus op het delen en integreren van informative tussen onafhankelijke knooppunten binnen coöperatieve netwerken, en het faciliteren van transparante toegang tot de informatie in hun databases, vormen de basis voor een efficient samenwerking. Het is echter duidelijk dat onafhankelijke knooppunten hun eigen informative heterogeen modelleren in hun database schema's. Voordat het uitwisselen van enige informative tussen deze databases kan plaatsvinden zal daarom een aantal uitdagingen aangegaan moeten worden, waaronder: (i) *identificatie en vaststelling van overeenkomsten tussen concepten gedefinieerd in onafhankelijke database schema's*, (ii) *oplossing van bestaande heterogeniteiten tussen database schema's* van de betrokken knooppunten, en (iii) *integratie van deze database schema's*. Over het algemeen zijn dit gecompliceerde taken, zelfs om met de hand uit te voeren, en deze moeilijkheidsgraad neemt toe met het aantal knooppunten in het federatieve netwerk, alsook met de grootte van de database schema's. Een aantal onderzoeksrichtingen en -prototypes heeft zich dan ook bezig gehouden met het automatiseren van het vergelijken van schema's, terwijl anderen zich onafhankelijk hiervan gericht hebben op het automatiseren van het integreren van schema's. Ondanks deze eerdere pogingen in zowel academische als commerciële omgevingen, vraagt het vergelijken en integreren van schema's vandaag de dag nog steeds een grote hoeveelheid handwerk en zijn er vele open onderzoeksvragen op deze gebieden.

Dit proefschrift beschrijft een *geautomatiseerde maar onder supervisie opererende gecombineerde benadering waarbij de concepten van het vergelijken en integreren van relationele database schema's aangepakt en samengevoegd worden*. De belangrijkste

[1] Vertalling door Leo Breebaart

contributie van dit proefschrift is derhalve een nieuwe gecombineerde schemavergelijkings/integratie benadering. Een *proof of concept* voor deze aanpak wordt gegeven door de implementatie van een prototype systeem genaamd SASMINT – Semi Automatic Schema Matching and INTegration. Het SASMINT systeem identificeert automatisch een groot aantal syntactische, semantische en structurele heterogeniteiten tussen relationele database schema's. Vervolgens probeert het deze heterogeniteiten op te lossen door de gebruiker ter validatie een lijst van mogelijke correspondenties tussen de vergeleken systemen voor te leggen. Het systeem genereert dan automatisch uit deze lijst een geïntegreerd schema.

Hoofdstuk 1 van dit proefschrift presenteert de motivatie voor dit werk, de primaire onderzoeksvragen, de doelstellingen, en de belangrijkste contributies van dit onderzoek.

Hoofdstuk 2 gaat dieper in op de basisdefinities en de classificatie van concepten uit de huidige *state of the art*, in relatie tot architecturen, methodes, en systemen die het delen en uitwisselen van gedistribueerde en heterogene informatie mogelijk maken.

Daar heterogeniteit centraal staat in de schemavergelijkings en -integratie processen, zijn identificatie en oplossing van verschillende types heterogeniteit cruciaal voor het succes van deze processen. Hoofdstuk 3 presenteert verschillende taxonomieën voor heterogeniteiten, en reduceert deze vervolgens tot database schema heterogeniteiten, het hoofdonderwerp van dit proefschrift.

Na in deze eerste drie hoofdstukken de basis gelegd te hebben voor ons onderzoek, behandelt de rest van dit proefschrift het ontwerp en de uitwerking van onze voorgestelde benadering. Als centraal hoofdstuk van dit proefschrift bevat Hoofdstuk 4 een literatuurstudie die zich richt op (i) database-integratie en -interoperabiliteit, (ii) database schemavergelijking, (iii) database schema-integratie en (iv) ontologievergelijking en -samenvoeging. Vervolgens introduceren wij onze voorgestelde oplossing, SASMINT, om een aantal benoemde open problemen aan te pakken. De SASMINT benadering verhoogt de nauwkeurigheid van de schemavergelijking door het gebruik van een gewogen combinative van een aantal schemavergelijkingsalgoritmes, waar elk algoritme een specifieke categorie syntactische, semantische of structurele conflicten aanpakt. Een andere bijdrage van SASMINT die in Hoofdstuk 4 wordt behandeld is de introductie van onze zogenaamde SAMPLER techniek, die voor elk doeldomein semiautomatisch de juiste gewichten bepaalt voor de algoritmes die toegepast zullen worden in het linguïstische vergelijkingsproces. Hoofdstuk 4 introduceert tevens een overzicht van de verzameling regels die automatische generatie mogelijk maakt van zowel de geïntegreerde schema's als van de afgeleide constructies die representatief zijn voor de geschiedenis van het integratieproces in het coöperatieve netwerk. Ook onze ontwikkeling van de SASMINT Derivation Markup Language (SDML), waarmee het creëren van zowel persistente schemavergelijkingsresultaten als persistente schema-integratieresultaten beschreven en ondersteund wordt, wordt behandeld in dit hoofdstuk.

Net als bij ieder ander onderzoek is het belangrijk om de door ons voorgestelde benadering te verifiëren en te valideren. Om dit te bewerkstelligen hebben wij een implementatie van onze aanpak ontwikkeld in de vorm van het SASMINT systeem, waar in Hoofdstuk 5 het focus op gericht is. In dit hoofdstuk worden zowel de belangrijkste onderdelen als de werking van SASMINT beschreven. Alhoewel naar onze mening het ondersteunen van gebruikersinteractie door middel van een GUI fundamenteel is voor het proces van semiautomatische schemavergelijking en -integratie, is dit typisch een component die ontbreekt in het onderzoeks- en ontwikkelwerk op dit gebied. Een GUI is nodig om de gebruiker te ondersteunen in het verifiëren van zowel de automatisch geïdentificeerde schemaconflicten en -correspondenties als van het voorgestelde geïntegreerde schema. Via SASMINTs GUI kunnen

gebruikers met het systeem interacteren, de juiste procesweegfactoren aangeven, de automatisch gegenereerde resultaten goedkeuren/veranderen/afkeuren, en de resultaten bewaren van zowel schemavergelijking als schema-integratie.

Om de resultaten van dit onderzoek te demonstreren en te evalueren en om de kwaliteit van het SASMINT systeem te meten, hebben wij een aantal experimenten uitgevoerd. In het bijzonder is de schemavergelijkende component van SASMINT vergeleken met, en geëvalueerd tegen een ander state of the art vergelijkingssysteem. De benadering van SASMINT is echter uniek in het feit dat het een fusie is van zowel schemavergelijkings- als schema-integratieprocessen. Het was daarom niet mogelijk een tegenhangen te vinden waarmee de schema-integratiecomponent van SASMINT vergeleken kon worden. In onze evaluatie-experimenten hebben we derhalve alleen maar gemeten hoe succesvol er correcte resultaten geproduceerd worden. Deze experimenten worden uitvoerig behandeld in Hoofdstuk 6 van dit proefschrift.

Dit proefschrift eindigt met een uitleg over hoe de primaire onderzoeksvragen zijn beantwoord.

Acknowledgments

Finally … I have gotten to reach the *happy* end ☺.

During the last couple of years, there have been times I thought I would not be able to finish my PhD. However, the stubborn side of me said that I would finish, which proved to be true eventually! This really makes me feel proud; especially considering that I had to go back to Turkey after spending two years as AiO in the Netherlands and having to complete the research from remote. Of course, this success does not only belong to me. It is now a pleasure to thank those who have contributed to this success.

First of all, I am truly thankful to my promoter, Hamideh Afsarmanesh, whose support, encouragement, and guidance from beginning to the end genuinely pushed me towards finishing my PhD. She has been an excellent promoter and strived to teach me how to do research and write academic papers. I really appreciate Hamideh for her confidence on myself and my research.

Furthermore, I owe my special thanks to Bob Hertzberger, for accepting me to the institute in the first place, and supporting my research even after I returned back to Turkey. I also would like to thank the evaluation committee members, Prof. Dr. Bob J. Wielinga, Prof. Dr. Lynda Hardman, Prof. Dr. Marian T. Bubak, Prof. Luis M. Camarinha-Matos, and Prof. Dr.-Ing. Bernhard R. Katzy for spending their time on my thesis and on the promotion activities.

I also thank to Erik Hitipeuw, Jacqueline van der Velde, Saskia van Loo, and Virginie Mes for their help and assistance during my PhD. Erik, I am grateful to all your help with the administrative tasks during the graduation process.

My thanks also go to my colleagues at UvA for their support. I first of all thank to former members of the COLNET group: Cesar Garita, who helped me at the beginning of my PhD and did courage me for writing my first publication. Ammar Benabdelkader, for helping me through my 'course assistant' period. Ersin Kaletaş, both for guiding me throughout the PhD, and also for the time we spent together during the coffee breaks. Thanks to Victor Guevara, for his help and advises and also for the nice chats. Thanks to Simon Msanjila and Ekaterina Ermilova for helping me after I got back to Turkey. Thanks to Naser Ayat and Jafar Tanha also, who are the newest members of the COLNET group.

I am also thankful to members of our 'Turkish gang' for making my life more pleasant at the Institute: Ersin, Hakan Yakalı, Çağkan Erbaş. Also thanks to my other Turkish friends in the Netherlands, Başak Kükrer and Selin Erbaş, for the fun we had.

I owe my deepest gratitude to Leo Breebaart for helping me with the translation of my English summary to Dutch.

I appreciate my friends who gave me support when writing my thesis. My special thanks to my friends Deniz Yazıcı and Yasemin Seydim for both encouraging me and also for reading

my thesis thoroughly. I would like to also thank my ex-home mate Almıla Kından, for her support and encouragement.

I would like to thank my mother Aynur, my father Necat, my brother Özgür, my sister-in-laws Canan and Cansu, and my niece Öykü, for all their support, help, and patience. My mother and father helped me a lot with taking care of my son as I was busy with the thesis. Thank you so much.

Last but not least, I thank my dear husband, İlker. We had very fruitful discussions all the times about my research subject. His love and support carried me through the rough times. And of course, my little son, Egemen, your existence always encouraged me to finish my PhD. I am sorry that you were told your mommy was at work, though I was working on my thesis in the next room. However, you will be proud of me when you are old enough to understand the meaning and the value of doing a PhD. *Sizi çok seviyorum*…