Schema Matching and Integration in Large Scale Scenarios

Khalid Saleem

Université Montpellier II
— Sciences et Techniques du Languedoc —

Thèse

pour obtenir le grade de
Docteur de l'Université Montpellier II

Discipline : Informatique
Spécialité Doctorale : *Informatique*
Ecole Doctorale : *Information, Structure, Systèmes*

présentée et soutenue publiquement par

Khalid Saleem (Candidate)
le Novembre 2008

Schema Matching and Integration in Large Scale Scenarios

Jury

Chantal Soule-Dupuy, Professeur, Université Toulouse I, ... Rapporteur
Patrick Valduriez, Directeur de Recherche, INRIA, ... Rapporteur
Mohand-Said Hacid , Professeur, Université Lyon I, ... Examinateur
Danièle Herin, Professeur, Université Montpellier II, ... Examinateur
Ela Hunt, Lecturer, Strathclyde University, Glasgow, ... Examinateur
Zohra Bellahsene, Professeur, Université Montpellier II, ... Directeur de Thèse

Abstract

Semantic matching of schemas in heterogeneous data sharing systems is time consuming and error prone. The dissertation presents a new robust automatic method which integrates a large set of domain specific schemas, represented as tree structures, based upon semantic correspondences among them. The method also creates the mappings from source schemas to the integrated schema. Secondly, the report gives an automatic technique to compute complex matchings between two schemas.

Existing mapping tools employ semi-automatic techniques for mapping two schemas at a time. In a large-scale scenario, where data sharing involves a large number of data sources, such techniques are not suitable. Semi-automatic matching requires user intervention to finalize a certain mapping. Although it provides the flexibilty to compute the best possible mapping but time performance wise abates the whole matching process. At first, the dissertation gives a detail discussion about the state of the art in schema matching. We summarize the deficiencies in the currently available tools and techniques for meeting the requirements of large scale schema matching scenarios. Our approach, PORSCHE (Performance ORiented SCHEma Mediation) is juxtaposed to these shortcomings and its advantages are highlighted with sound experimental support.

PORSCHE associated algorithms, first cluster the tree nodes based on linguistic label similarity. Then, it applies a tree mining technique using node ranks calculated during depth-first traversal. This minimises the target node search space and improves time performance, which makes the technique suitable for large scale data sharing. PORSCHE implements a hybrid approach, which also in parallel, incrementally creates an integrated schema encompassing all schema trees, and defines mappings from the contributing schemas to the integrated schema. The approach discovers 1:1 mappings for integration and mediation purposes. Formal experiments on real and synthetic data sets show that PORSCHE is scalable in time performance for large scale scenarios. The quality of mappings and integrity of the integrated schema is also verified by the experimental evaluation.

Moreover, we present a technique for discovering complex match (1:n, n:1 and n:m), CMPV (Complex Match Proposition and Validation), between two schemas, validated by mini-taxonomies. The complex match proposition part is an extended version of schema matching part of PORSCHE. The mini-taxonomies are extracted from the large set of domain specific metadata instances represented as tree structures. We propose a framework, called ExSTax (Extracting Structurally Coherent

Mini-Taxonomies) based on frequent sub-tree mining, to support our idea. It is the extension of the tree mining method of PORSCHE. We further utilise the ExSTax technique for extracting a reliable domain specific taxonomy.

Keywords
Data interoperability, XML schema tree, schema matching, schema mapping, schema mediation, tree mining, large scale, ontology learning, mini-taxonomies, collaborative ontology construction.

TITRE en français : **Intégration de Schémas Large Echelle**

Resumé

La mise en correspondance sémantique appliquée à des schémas hétérogènes dans les systèmes de partage de données est une tache fastidieuse et source d'erreurs. La thèse présente une nouvelle méthode automatique et robuste qui intègre un grand nombre de schémas sous forme arborescente et de domaine spécifique. Elle permet de découvrir des correspondances sémantiques entre eux. La méthode crée également les mappings entre des schémas sources et le schéma intégré. Puis, le manuscrit présente une technique pour découvrir d'une manière automatique des correspondances complexes entre deux schémas.

Les outils de mise en correspondance existants utilisent des techniques semi-automatiques uniquement entre deux schémas. Dans un scénario à grande échelle, où le partage des données implique un grand nombre de sources de données, ces techniques ne sont pas adaptées. De plus, la mise en correspondance semi-automatique nécessite l'intervention de l'utilisateur pour finaliser les mappings. Bien qu'elle offre la possibilité de découvrir les mappings les plus appropriés, les performances s'en trouvent fortement dégradées. Dans un premier temps, le manuscrit présente en détails l'état de l'art sur la mise en correspondance. Nous expliquons les inconvénients des outils actuellement disponibles pour répondre aux contraintes d'un scénario à grande échelle. Notre approche, PORSCHE (**P**erformance **OR**iented **SCHE**ma Mediation) évite ces inconvénients et ses avantages sont mis en évidence de manière empirique.

Le principe de l'algorithme de PORSCHE consiste à regrouper d'abord les nIJuds de l'arbre selon la similarité linguistique de leurs labels. Ensuite, des techniques de fouilles d'arbres utilisant les rangs des nIJuds calculés au moyen du parcours en profondeur de l'arbre sont appliquées. Cela réduit l'espace de recherche d'un nIJud cible et améliore par conséquent les performances, ce qui en fait une technique adaptée au contexte large échelle. PORSCHE implémente une approche hybride, qui crée également en parallèle et de manière incrémentale un schéma intégré qui englobe tous les schémas, tout en définissant les correspondances entre ces derniers et le schéma intégré. L'approche découvre des correspondances 1:1 dans un but d'intégration et de médiation. Finalement, des expérimentations sur des jeux de données réels et synthétiques montrent que PORSCHE passe à l'échelle avec de scénarios de grande échelle. La qualité des correspondances découvertes et l'intégrité du schéma intégré sont également vérifiées par une évaluation empirique.

Par ailleurs, nous présentons une technique CMPV (**C**omplex **M**atch **P**roposition et **V**alidation), pour la découverte de correspondances complexes (1:n, n:1 et n:m),

entre deux schémas, validée par l'utilisation de mini-taxonomies. Cette partie est une version étendue de l'aspect de mise en correspondance de PORSCHE. Les mini-taxonomies sont extraites d'un vaste ensemble de métadonnées de domaine spécifique représenté comme des structures arborescentes. Nous proposons un cadre, appelé ExSTax (**Ex**tracting **S**tructurally Coherent Mini-**Tax**onomies) basé sur la fouille d'arbres pour appuyer notre idée. C'est l'extension de la méthode fouille d'arbres de PORSCHE. Enfin, on utilise la technique ExSTax pour extraire une taxonomie fiable spécifique à un domaine.

MOT-CLES

Interopérabilité des données, schéma XML sous forme arborescente, mis en correspondance de schémas, mapping, intégration de schéma, fouille d'arbres, grande échelle, apprentissage d'ontologie, mini-taxonomies, construction d'ontologies collaboratives.

DISCIPLINE

INFORMATIQUE

INTITULE ET ADRESSE DE L'U.F.R. OU DU LABORATOIRE

Le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier
LIRMM, UMR 5506 - CC 477, 161 rue Ada, 34392 Montpellier Cedex 5 - France

To my family

Acknowledgements

First of all, I want to express my gratitude to my parents for their encouragment and prayers for all these years. Their support to pursue this goal had been enormous.

I wish to thank my supervisor Dr. Zohra Bellahsene for being my mentor in research. Her consistant confidence in me to publish research had propelled me to where I stand today.

I would like to thank all my thesis jury members. Special thanks to Dr. Chantal Soule-Dupuy and Dr. Patrick Valduriez for their valuable comments and remarks as reviewers of the dissertation. I am indebted to Dr. Mohand-Said Hacid, Dr. Danièle Herin and Dr. Ela Hunt for taking out time from their hectic schedules, to act as examiners for my work. I am also very grateful to Dr. Ela Hunt for her critical analysis of our research publications while her stay at ETH, Zurich as a Marie Curie research fellow.

Special thanks to my friends and collegues at LIRMM, Federico Del Razo, Cheddy Raissi, Marc Plantevit, Fabien Duchateu, John Tranier, Rémi Coletta and Robbie Coenmans for their informal discussions and support.

My stay in France for persuing the doctorate degree has been financially supported by the Higher Education Commission, Governement of Pakistan, in conjunction with Government of France.

The acknowledgements section would never be complete, if I do not acknowledge the support of my family, who missed me at dinners and evening walks, while I was at laboratory working late...

Montpellier, Novembre 2008

Khalid Saleem

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Schema matching is the process, which finds correspondences between elements of two schemas. This dissertation studies the problem of large scale schema matching and integration. Large scale scenarios can be categorized according to the input set of schemas as (i) two large schemas or (ii) a large set of schemes. In our work schemas are considered as tree structures for exploiting the contextual similarity among them. It allows us to design a generic solution and utilize the tree mining technique to tackle the large scale aspect. XML schema is a good example, which possesses the tree structure.

In literature [25] several variations of schema matching problem have been researched as ontology matching, ontology alignment, schema reconciliation, mapping discovery, representation matching, discovering semantic correspondence.

1.1 Schema Matching and Integration

The word schema has its origin in Greek, meaning "shape" or "plan". From computer science perspective it is defined as the description of the relationship of data/ information in some structured way or a set of rules defining the relationship.

Schema matching can be defined as *the process of finding similarities between two schemas*. To support this process, different types of schema related information can be exploited. For example schema structure, schema elements' names and associated data types and domain ranges over these data types or data instances etc. Schema matching is the core activity in the integration of two or more schemas which has a broad application context as discussed in section 1.2. Figure 1.1 presents one such scenario.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Abbeys>
    <author />
    <titlekeywords />
    <parentcategory />
    <pricerange />
    <isbn />
    <pubdate_option />
</Abbeys>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<AmericanBookCenter>
    <searchcrit1 />
    <searchcrit2 />
    <published>
        <datefrom />
        <datetill />
    </published>
    <onderwerp />
    <sel_publisher />
</AmericanBookCenter>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<Bookbeat>
    <author />
    <catalog />
    <title />
    <keyword />
</Bookbeat>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<Dymocks>
    <Title />
    <Author />
    <Department />
    <ISBN />
</Dymocks>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<bookery>
    <lastname />
    <firstname />
    <Title />
    <description />
    <categoryID />
</bookery>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<Book-Place>
    <ctitle />
    <CAUTHOR />
    <keyword />
    <SEARCH_TEXT />
    <IMPRINT />
    <SearchFilters>
        <FILTER_binding_code />
        <FILTER_reader_code />
    </SearchFilters>
</Book-Place>
```

Figure 1.1: XML schema instances of different web based query interfaces for book searching.

Example 1.1 Querying Web Interfaces: Let us consider an example of a person searching for good deals for books, for reading in her next holidays. There are hundreds of web interfaces available for query purposes. She can not query each interface and then compare each query result. The best answer to his problem would be a virtual mediated interface which is mapped to each of the physical interfaces over the web in the books domain. The results from each query are in turn integrated and displayed according to her preferences. The first step in the implementation of the virtual interface is the matching of all possible/ available interfaces over the web in the specified domain. Once the mappings between the individual interfaces and the integrated interface has been done, query processing and results display processes can be initiated. The mediated schema with mappings can be cached for future utilisation by other users with similar requirements. The web query interfaces follow a hierarchical tree like structure as shown in figure 1.1 for books domain taken from TEL-8 dataset [1].•

[1] http://metaquerier.cs.uiuc.edu/repository

We have developed a framework *P*erformance *Or*iented *Sche*ma Mediation (discussed in chapter 5), having good time performance results with approximate schema mappings from source to integrated schema. PORSCHE considers schemas as tree structures. It employs a tree mining data structure (sub-trees of size one) based on depth-first ordering of nodes, to find matchings among schemas for integration and mediation. For the integration purposes simple matchings are considered. Our second research work revolved around finding a solution for complex match discovery. We extracted the matching part of PORSCHE and extended it to discover complex matches between two schemas, *C*omplex *M*atch *P*roposition and *V*alidation (discussed in chapter 7). To further support the discovered complex match, we used mini-taxonomies representing domain concepts to validate the discovered complex matches. We extended the tree-mining part of PORSCHE, to find frequent growing sub-tree patterns in the large set of given schemas trees (ExSTax technique discussed in chapter 6). The resulting frequent sub-trees are considered to be the mini-taxonomies.

1.2 Application Context of Large Scale Schema Matching

Over the years technology has made this world a web of digital information, where digital systems are appearing at an exponential rate. At individual level, personal or professional, or organisational level, there exists an unending list of digital devices cooperating together to solve problems. Every day a new gadget hits the market, creating a ripple-effect in its surrounding operating environment and giving rise to new innovations in the field around it. The collaboration between these devices eventuates in better performance and results. For us, the database people, this is emergence of new form of data or information, which has to be utilised in the most efficient and effective manner. The ability to exchange and use of data/information between different devices (physical or logical), is the basic activity in any type of system, usually referred to as data interoperability [84]. Thus the domain of data interoperability has also evolved with emergence of new devices and systems.

1.2.1 Autonomy, Distribution and Heterogeneity Aspects of Data Sources

Data source can be defined as anything from which data originates. It can be a database within a DBMS, a simple text file or any other emerging new device

as discussed in preceeding section. Every data source has to know the meaning encoded in the data (personal or of some other data source). It can be learned primarily from the *Schema*, representing the data of the data source. For inception of a system there are different levels, and each level can have its own description. For example, in relational database systems, a database schema is a set of relations and attributes. Thus for an application, a schema gives the best way to understand the semantics of the underlying data instances.

Today, the whole scenario related to data sources is quoted as "data everywhere" [37]. And the participating data sources are presenting more and more autonomous behaviour, with a much greater will to share their data. Some of the best examples are the social and P2P network environments. Future implementations are weighing for a broader virtul network platforms where individual environments like Facebook (social network) can exchange data with bittorrent (P2P network). So, we are going to encounter one of the biggest collaborating, autonomous, distributed and heterogenic data environment.

1.2.2 Large Scale Scenarios

Schema matching is a basic task in almost every data intensive application in the evergrowing dataspaces. The central platform being utilised to share the digital information is the world wide web, which is evolving from an unstructured data presenter to a more semantically structured and reasoning entity, termed as semantic web. Semantic web provides a framework where machines can move one step further, understand the structures of data, the contextual meaning of the data and autonomously reason over it. Thus providing the ground for semantically richer applications for large scale distributed information systems. Secondly, it is pushing the schema matching research to utilize the processing power not available in the past and directly increasing the industry investment proportion in the matching domain [21][2]. In the recent years, several applications requiring large scale semantic matching have emerged. These have been categorized as information sharing, processing or delivering systems.

In the information sharing domain we have seen peer-to-peer (P2P) database systems and the social network environments as the leading application development and investment gainers. P2P started as simple file sharing application. Recently, the schema based architectures are opening ways to share information between different P2P networks. Similarly, initial social networks used simple user oriented tagging of

[2]Markets for semantic technology products and services will grow 10-fold from 2006 to 2010 to more than 50 billion dollars worldwide

shared resources. And today these environments are adopting OWL based ontologies following FOAF [3] standards.

Multi-agent and web services based systems represent the processing category in schema matching domain. Multi-agent systems try to solve problems which are difficult to be solved by one agent and require quick response, like online trading and disaster management systems [91]. Agents have their own ontologies to process the information, with the ability to learn and evolve over time. Several approaches have been researched to handle web services composition and discovery; WSML [4] is one of the standards, based on XML framework, for describing a web service. Similar to multi-agent systems, web services are used to solve large problems. A single service request can require several other inter-related services to be discovered and executed.

The core purpose of web is to deliver information as a result of user query. Web search engines [5], web interface query forms for backend databases [17], web mashups [6] are the examples of delivering systems. In these systems, information resources are compared with the requested query, and results are presented to user.

Motivation

We have been motivated by these scenarios, to find a generic time performance oriented solution for matching large number of data sources modeled as schemas, which can be large within themselves. As we have seen in the above application domains, we also need to integrate the data sources' schemas based upon the discovered matchings and create mappings from the sources schemas to the integrated schema.

1.3 Objectives of the Dissertation

The central goal of the dissertation is that, for schema matching and integration problem in the large scale scenario, we design an almost automatic technique with approximate match quality and good time performance.

Explicitly, the objectives are:

- Analyse the state of the art in schema matching and integration, to learn the available solutions and their applicability.

[3] Friend Of A Friend (FOAF) http://www.foaf-project.org
[4] Web Service Modeling Language (WSML) http://www.w3.org/Submission/WSML/
[5] http://swse.deri.org
[6] http://www.programmableWeb.com

- Outline the desiderata for the large scale schema matching and integration scenarios.
- Develop a framework for schema matching (simple and complex) and integration with almost no user intervention in the matching phase and good time performance.
- Provide within the framework, an apposite solution to handle different schema based data models like XML schema, relational data base etc.
- Design the solution, which can give good quality schema matching with minimum schema information, e.g., only the schema attributes names and the hierarchical relationship between the attributes within the schema.

1.4 Contributions

In this section we explicitly outline our contributions to fulfill the above mentioned objectives.

- The main contribution of this dissertation is our novel approach PORSCHE (**P**erformance **OR**iented **SCHE**ma Mediation). The approach is almost automatic and hybrid in nature. It is based on a tree mining technique supporting large scale schema matching and integration. To support tree mining, we model schemas as rooted ordered (depth-first) labelled trees. The idea provides a generic solution for any schema model which possesses a hierarchical structure.

  The approach employs node level clustering, based on node label similarity, to minimise the target search space, as the source node and candidate target nodes are in the same cluster. Label similarity is computed using tokenisation and token level synonym and abbreviation translation tables. The technique extends the tree mining data structure proposed in [104]. It uses ancestor/ descendant scope properties (integer logical operations) on schema nodes to enable fast calculation of contextual (hierarchical) similarity between them. The output is the mediated schema and a set of simple mappings from input source schemas to the mediated (integrated) schema and vice versa.

  The approach was implemented as a prototype. We report on experiments using different real (OAGIS[7], xCBL[8]) and synthetic scenarios, demonstrating:

[7] http://www.openapplications.org
[8] http://www.xcbl.org

a) high performance for different large scale data sets, which shows that our method is scalable and supports a large scale data integration scenario;

b) input schema selection options (smallest, largest or random) for the creation of initial mediated schema, allowing us to influence matching performance;

c) quality evaluation using *precision, recall and F-measure* as measures of mapping quality;

d) analysis of the integrity of integrated schema with reference to completeness and minimality measures;

e) quadratic time complexity of the PORSCHE algorithms.

- We further exploit the tree mining technique to extract mini-taxonomies, as domain concepts, from the large scale input, and call it ExSTax(Extraction of Structurally Coherent Mini-Taxonomies). The technique builds clusters of similar terms based upon labels similarity of input schemas' elements. The similarity is computed using label's syntactic, lexical and contextual (hierarchical) occurrence in the schema as in PORSCHE. Each cluster is represented by a single symbol i.e., the most frequent label in the input set of schemas, in each cluster. The tree mining algorithm identifies the frequent sub-tree patterns as the mini-taxonomies. We report on experiments using different real (COURSES [9]) and synthetic scenarios, demonstrating quality of generated mini-taxonomies using precision measure.

- We demonstrate the use of the automatically generated mini-taxonomies to solve complex match problem between two schemas. The method proposes semantically approximate complex mappings as 1:n (leaf node to non-leaf node), n:1 (non-leaf node to leaf node) and n:m (non-leaf node to non-leaf node) using the simple mapping technique employed in PORSCHE. Then it utilises the automatically extracted mini- taxonomies for the validation of the proposed mappings. Non-leaf node implies a set of leaf nodes of subtree rooted at the non-leaf node.

- ExSTax approach is further extended to automatically produce a trustable basic domain taxonomy from the given set of domain specific schemas, implying domain community consensus over it.

[9] http://www.cise.ufl.edu/research/dbintegrate/thalia/

- Finally, the dissertation evince the relationship between the basic schema matching techniques, research domains utilizing these techniques and the application domains which benefit and propel this research. We propose a taxonomy of schema matching strategies with respect to large scale scenario.

1.5 Structure of the Dissertation

This dissertation is organised into 8 chapters. Current chapter introduced the problem of schema matching and its application domain in large scale scenarios. Chapter 2 outlines the problem definition for the large scale schema matching and integration. In chapter 3, we give a detail account of state of the art in schema matching and integration. Chapter 4 discusses the desiderata for solving the large scale schema matching and integration problem. We present our approach (PORSCHE) for element level simple schema matching and integration in Chapter 5. In chapter 6 we discuss the mini-taxonomies generation and its validity. Our complex schema matching technique and related evaluation is discussed in chapter 7. Conclusions and future perspective of our research is given in chapter 8.

Chapter 2

Problem Definition

This chapter defines the main schema integration and mediation problem in the large scale scenario, with reference to autonomy, distribution and heterogeneity [82] aspects of schemas. First, we give the specific problems of schema integration, simple schema matching and mediation based on the matchings, and secondly, we address the problem of complex matching between two schemas. One of the goals of the dissertation is to solve the schema matching problem, by inspecting minimum details about the input schemas, with maximum automation. Therefore, we try to exploit the schema elements names and the hierarchical structure of the elements to find the matches. First, we discuss the notion of tree structure in the schema models to aid us design a generic approach for solving the problem. This helps in addressing the heterogeneity of the schemas.

2.1 Schema Trees

A schema defines a model to represent the data. There are several schema models available e.g., relational, XML schema, ontology, object oriented representation, web interface query form schema etc. Most of the models posses the hierarchical structure and some can be converted to hierarchical structure [36, 18, 57].

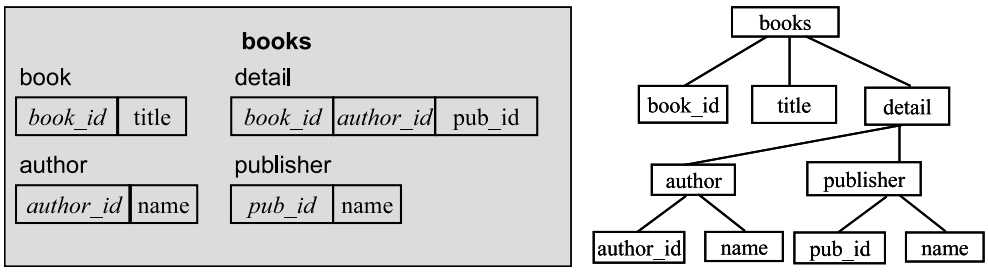

Figure 2.1: Relational database about books and corresponding schema tree.

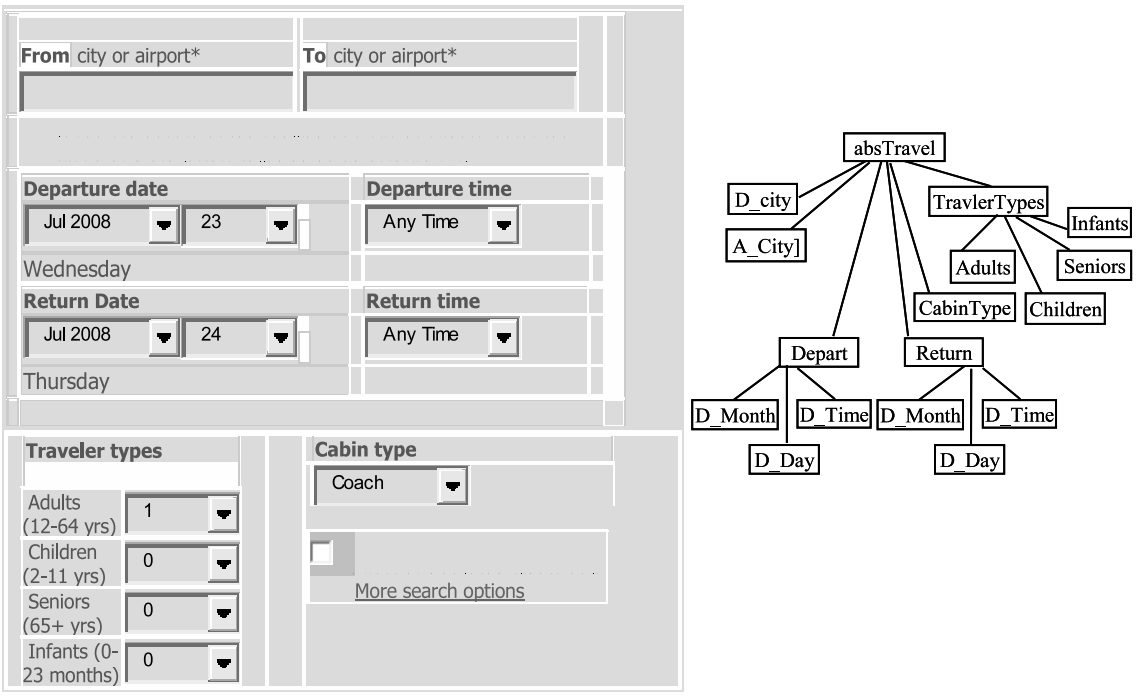

Figure 2.2: ABS Travel web interface form and corresponding schema tree.

The tree structure for a data model inherently supports the contextual meanings of the descendant nodes, thus making the matching process semantically more viable. Tree structure in figure 2.1 shows the difference between the two nodes, the **author** *name* and the **publisher** *name* with respect to their parent nodes. Secondly, developing algorithms for matching can depend upon recursive or incremental methods for fast calculation of matches. For example, top-down approach with depth-first traversal can help one match the parent node before the children, thus defining the context for the children nodes. Thirdly, it can seamlessly support the expansion scenario in the integration process. Non-existent similar nodes in target schema for a certain source schema node, can be added as child node to the already matched parent node.

Keeping in view the above mentioned features of tree structure, we propose a schema to be treated as a tree structure. Following is the related definition:

Definition 2.1 (Schema Tree): A schema $S = (V, E)$ is a rooted, labelled tree[104], consisting of nodes $V = \{0, 1, \ldots, n\}$, and edges $E = \{(x, y) \mid \text{x,y} \in V\}$. One distinguished node $r \in V$ is called the root, and for all $x \in V$, there is a unique path from r to x. Further, lab:V $\rightarrow$ L is a labelling function mapping nodes to labels in $L = \{l_1, l_2, \ldots\}$.

In the subsequent section we present the problem definitions for our work.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="arizonaCourses">
        <xs:annotation>
            <xs:documentation>University of Arizona</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Course" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence minOccurs="0" maxOccurs="unbounded">
                                <xs:element name="Time" type="xs:string" minOccurs="0"/>
                                <xs:element name="Day" type="xs:string" minOccurs="0"/>
                                <xs:element name="Place" type="xs:string" minOccurs="0"/>
                                <xs:element name="Instructor" type="xs:string" minOccurs="0"/>
                        </xs:sequence>
                        <xs:attribute name="Code" type="xs:string" use="optional"/>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

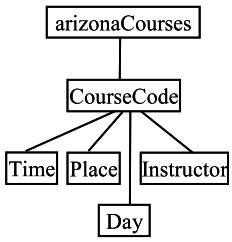

Figure 2.3: Arizona University Courses XML schema and corresponding schema tree.

2.2 Schema Integration and Mediation

The most general approach for a data integration system is proposed in [62] as a combination of a global schema, the sources, and the mapping from the sources to the global schema. We use the word mediated schema instead of global schema. Based on this approach we formally define the problem as:

Problem 1 (Schema Integration and Mediation): Given a set of schemas S_1, S_2,... S_u, discover a schema S_m, which holds all concepts, existing in the set of u schemas. And for each element s of S_i where $1 \leq i \leq u$, create a mapping to the most appropriate semantically matching element s_m of S_m.

Keeping in view the above problem we try to devise a system which can perform well and abide by the large scale, heterogenic and semantic aspects (discussed in next chapter) of todays distributed federated information systems. To handle these aspects one needs to suggest solutions which can handle the diversity of the data sources. To propose a uniform solution in this regard, we embarked upon the hierarchical nature of data sources as discussed in the previous section. In next subsections we divide this problem into sub-problems of schema integration, schema matching and mediation, and complex matching.

2.2.1 Schema Integration

For creating an integrated system, one has to have an integrated schema encompassing all concepts of the source schemas. In research [5], different techniques have been defined for creating an integrated schema incremental or holistic. Since we are targeting the large scale scenario, one has to define the problem with minimum pruning of the sources. Therefore, we take up the problem as an incremental schema integration approach, supporting the matching and mediation features.

For an incremental schema integration approach, one has to have an initial seed schema or the initial mediated schema. The system searches for concepts from each input schema in the mediated schema, and concepts not found are added to the mediated schema. With regard to initial mediated schema, we propose three possible assumptions:

1. Initial mediated schema can be the smallest schema in size among the input schemas.

2. Initial mediated schema can be the largest schema in size among the input schemas.

3. Initial mediated schema can be any randomly selected schema among the input schemas.

Our hypothesis regarding assumptions is that while creating the integrated schema; assumption 1 supports more flexibility in extending the initial mediated schema and assumption 2 give the advantage of minimum number of concepts added to the mediated schema thus improving the over all time performance in large scale scenario.

Problem 2 (Schema Integration): Given a set of schemas S_1, S_2,... S_u, select one of the schema as the initial mediated schema $S_{mInitial}$. And then merge each schema S_i where $1 \leq i \leq u$ to the schema $S_{mInitial}$.

Merge process requires the matching of elements from source schema to the initial mediated schema, which is discussed in next sub-section.

2.2.2 Simple Schema Matching and Mediation

In literature, there have been several formal definitions of schema matching problem. Our approach for schema matching problem is an extension of the research described in [86]. The problem is defined from the perspective of a schema integration perspective.

Problem 3 (Simple Schema Matching): Given two schemas S_{source} and $S_{mediated}$, for each element s_s of S_{source}, discover the most appropriate, semantically matching, element s_m of $S_{mediated}$.

The problem is also called 1:1 matching. It focuses on discovering semantic matches like *author* is similar to *writer*. The problem emphasizes the discovery of possible similar matches for a source schema element in the mediated schema, which can be 0 or more. In case of more than one semantically similar elements in target mediated schema, one of the matches has to be selected as the final match. In literature, schema matching problem definition is also supported by the assumption that the problem has to be solved by utilising all available information. It can be categorised as schema description, data instances, constraints, previous match results, heuristics and user input. Since we are tackling the problem in the large scale scenario, we assume the input for the problem is just the schema elements' labels and their corresponding tree structures.

Problem 3 emphasizes existing matches from the source to target schemas. In the integration scenario, this definition has to be further extended by the mediation aspect. From mediation one means to say, there must exist a mapping for each element of source schema in the mediated schema. This can only be fulfilled, if the semantically similar elements for source schema elements are non-existent in mediated schema, and the source schema elements must be added to the mediated schema. With reference to the definition of integrated system requirements given in [62], we present the schema mediation problem as:

Problem 4 (Schema Mediation): Given two schemas S_{source} and $S_{mediated}$, create a mapping for each element s_s of S_{source} to the most appropriate, semantically matching, element s_m of $S_{mediated}$.

This problem definition emphatically covers the mediation aspect required in a schema integration and mediation system. And it also subsumes the problem 3 of simple schema matching. We cover the solution to this problem in chapter 5.

2.2.3 Complex Schema Matching

Complex schema matching covers the match cardinality aspect where more than one element from any source or target schema, participate in one matching. It is designated as 1:n, n:1 and n:m schema matching. The 1:n and n:1 matching can be

discovered with the help of element level matching but the n:m schema matching requires the schemas' structural metadata also [86]. The complex matching scenario covers two matching aspects; first, discovery of n elements in a match and second how these n elements participate in the match. For example, an element *price* from source schema is matched to a combination of two elements in target schema as *(price*tax)*.

Problem 5 (Complex Schema Matching): Given two schemas S_{source} and S_{target}, each representing a set of concepts C_{source} and C_{target} respectively. Each concept c belonging to C_{source} or C_{target} is represented by set of elements of the respective schema.
(i) Discover for each concept c_{source} of schema S_{source}, represented by the set of n elements $\{s_{s1}, s_{s2}, \ldots, s_{sn}\}$, semantically similar concept c_{target} of schema S_{target}, represented by the set of m elements $\{s_{t1}, s_{t2}, \ldots, s_{tm}\}$ where $\{n, m \geq 1\}$.
(ii) Let O=$\{o_1, o_2, \ldots, o_j\}$ be a set of operators and R=$\{r_1, r_2, \ldots, r_k\}$ be a set of rules for applying these operators. Discover the expressions which bind together the n elements for representing c_{source} and m elements for representing c_{target} using O and R.

We present an approach for solving the first part of the problem 5 in chapter 6.

2.3 Conclusion

In this chapter we have outlined the several problems in schema matching and integration scenario. We had first defined the main problem of schema integration and mediation, and then presented the sub-problems of schema integration, simple element matching and mediation and complex schema matching, which make up the basic research problem targeted in this report. We have also proposed the use of tree representation of schemas for exploiting the structural matching of schemas. In next chapter we present the state of the art in schema matching and integration with reference to the outlined research problems.

Chapter 3

State of the Art in Schema Matching and Integration

In this chapter we discuss the state of the art in schema matching and integration research. Previous work on schema matching was developed in the context of schema translation and integration [9, 24, 48], knowledge representation [43, 93], machine learning, information retrieval [27] and multidatabase management systems [98]. All these approaches aimed to provide a *good* quality matching but require significant human intervention [9, 27, 24, 43, 48, 64, 66]. However, they missed to consider the performance aspect, which is equally important in a large scale scenario (large schema or a large number of schemas to be matched).

Our target in this chapter is to present a classification of the schema matching research from its application point of view. We highlight this aspect because of the mushrooming and the dynamic nature of the large scale data intensive applications. Indeed, evolving large scale distributed information systems are further pushing the schema matching research to utilize the processing power not available in the past and directly increasing the industry investment proportion in the matching domain. This chapter reviews the latest application domains in which schema matching is being utilized. It also gives a detailed insight about the desiderata for schema matching and integration in the large scale scenarios. Another panorama which is covered by the survey is the shift from manual to automatic schema matching. Finally we present the state of the art in large scale schema matching, classifying the tools and prototypes according to their input, output and execution strategies and algorithms.

3.1 Schema Heterogeneity

Before we discuss the schema matching, we would present a synopsis of its requirement in the database world. Initially database or data source concept was considered to be a single centralised entity. But with the passage of time and advancement of technology, data sources within a system adopted the distribution property. Thus, in early eighties, giving rise to distributed database environments (DDE): a single system catering for numerous data sources, physically apart, connected through a network. The autonomy and heterogeneity level of the contributing data sources also increased in parallel. As a result we were presented with the Multidatabase Systems. Such a system has a global schema which mediates with the schemas of the independent data sources (autonomy), making up the system. A very detail discussion about the placement of different types of DBMSs, with reference to the heterogeneity, autonomy and distribution of data sources is given in [82]. In Figure 3.1, we present a simplified view (extracted from [82]) of the three important dimensions regarding distributed database management systems.

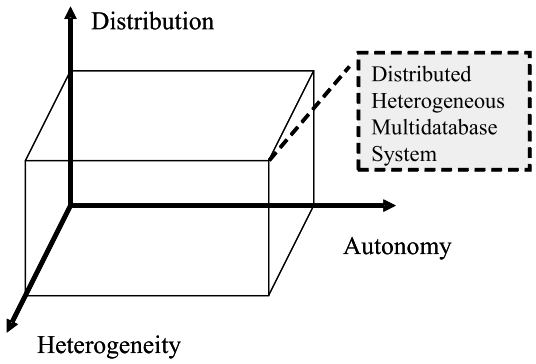

Figure 3.1: Placement of Distributed Heterogeneous Multidatabase System with reference to Heterogeneity, Distribution and Autonomy aspects

Today, with the ever growing deep web and semantic web in the focus, Distributed Heterogeneous Multidatabase System (DHMS) seems to be the most related data sources management system. Although not referenced directly, the concept of *Dataspaces* and related Dataspace Support Platform (DSP) in [37] show very similar characteristics to that of DHMS of the nineties. so, we have a transition from DHMS to DSP, both requiring the same level of mediation between heterogeneous schemas.

As discussed in the start of the chapter, previous implementations for mediation process required human intervention. A specific example is work authored in [98], presenting a fine architecture to scale the heterogeneity aspect of distributed

database systems using mediators, manually defined, and wrapper components. Today, the heterogeneity scale, governed by deep web and concepts like dataspaces require an automatic approach to mediate between the schemas of the contributing data sources. It is just out of the reach of manual implementation.

3.2 Revisiting the Schema Matching Problem

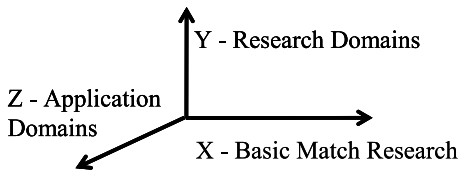

Figure 3.2: Schema Matching Dimensions

In this section, we present a classification of schema matching problem along three dimensions; basic match research, related research domains in schema matching and application domains dependent upon data interoperability. We focus on each dimension and show how these dimensions are interlinked, with the help of an example. Further, we give an overview of the research domain of large scale schema matching.

3.2.1 The Three Dimensions of Schema Matching

Schema matching has been researched from various perspectives by researchers belonging to different research and application domains. While reviewing the numerous approaches and techniques, we came to understand that schema matching is related to three different interlinked dimensions: (i) *basic match research*, related (ii) *research domains* and (iii) *application domains*. The three dimensions (Figure 3.2) are related as: algorithms are developed for *Basic Match Techniques* for exploiting some *Research Domain*, which is in turn responsible to carry out the objective of a certain *Application Domain*.

Considering the example in figure 3.3, we can state the link between the three dimensions as: (i) Basic match techniques applied to query interface attributes (based on attribute label, default value, data type, list of available values etc.) and their structural aspects, (ii) for query interface forms schema integration and mediation, (iii) in the application domain of querying the web based books resources.

Knowledge extraction for schema matching is done by exploiting two entities, (i) data and (ii) schema; structure describing the data. The availability of the two

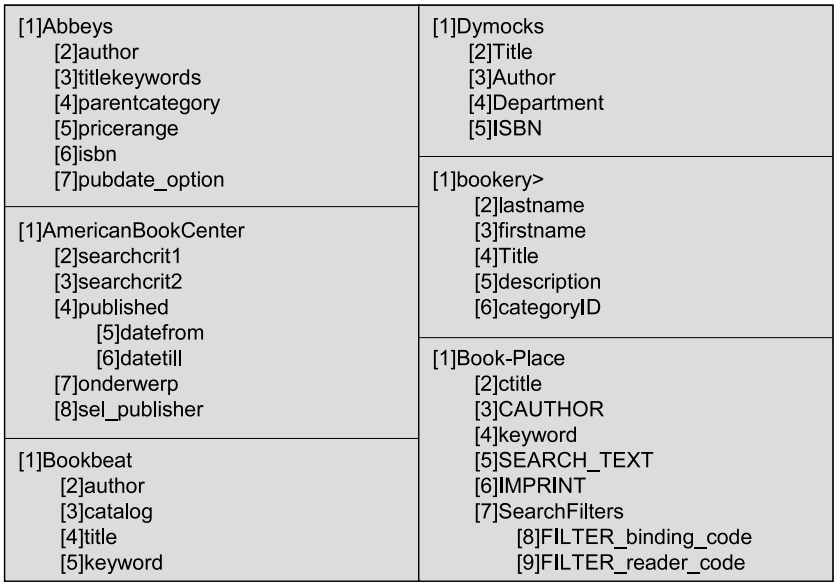

Figure 3.3: Hierarchical representation of query interfaces over the web for Books domain

entities is governed by the application specific constraints. For example *data security*, where direct data access is restricted and only controlled access through schema is granted. There are large number of techniques researched for schema matching with respect to data instances and schemas (section 3.4).

Table 3.1: Dimensions of Schema Matching - Basic Match Research

X - Basic Match Research	**Level**
Linguistic based	Element
Constraints based	Element
Graph based	Structure
Data Instance/Machine Learning based	Element/Structure
Use of External Oracle	Element/Structure

Basic match techniques (table 3.1)exploit the granularity aspect of a schema. It can be seen as the match algorithm development process for a certain entity in the schema. The entity can be the most basic constituent of schema e.g., field in a relational database schema table [7, 12], or the whole schema structure itself exploited, using some graph match techniques [24, 66, 75]. A combination of basic match techniques are utilized to resolve problems indicated in Table 3.2. Detail discussion on these techniques is given in section 3.4.

In **research domain** (data interoperability) (table 3.2), *Schema Integration* [5] can follow three possible approaches (i) *binary incremental* : two schemas are integrated at a time, following an upward binary tree pattern, (ii) *clustering incremental*

: clusters of schemas are created based upon some similarity function, an integrated schema is generated for each cluster and the resulting schemas are further grouped into clusters and so on [89, 60], or (iii) *holistic* : all schemas are pruned together and integrated [50] .

Table 3.2: Dimensions of Schema Matching - Research Domains

Y - Research Domains	**Type**
Schema Integration	Binary/Holistic
Schema Mapping Generation	Cardinality
Schema/ Mapping Evolution	
Ontology Alignment	Binary/Holistic
Match Quality Evaluation	

While comparing schemas, there is high probability that source element can have more than one matches in the target schema. One match has to be ranked the best manually or automatically, for the mapping purpose. The cardinality of mapping [86] demonstrates the numeric relationship of element correspondences. Semantically speaking, it is the number (combination) of elements in each of the two schemas, representing the same concept; 1:1, 1:n, n:1 and n:m element map cardinality.

The temporal changes to a schema and its effects on the existing mappings also provide another research domain. Since the web is an evolving entity, the *schema evolution* and related *mapping evolution* require attention. Methods like domain level corpus based schema matching [65] demonstrate how to maintain a repository of schemas, concepts representations and related mappings for subsequent handling of temporal changes in the constituent schemas of the domain. In another research work [99], the authors show how the changes in a schema are used to rewrite the queries representing the mappings. The research benefits from CLIO [52] which generates queries as the mappings expressions.

The ontology concept has been around since the early 90s. Today it is vigorously used in different applications requiring interoperability. Ontologies are used for knowledge representation and are similar to schemas to a certain extent; as both describe data domain with the help of terms with constrained meanings [93]. Techniques used for schema matching have been tailored for ontology matching (ontology alignment) [35].

Another similar research area, which has emerged as a by product of research in agents communication, is the tracking of changes in the source ontologies of agents called *ontology evolution* [81]. Since agents are independent entities, following their

own rules, they require different techniques for comparing and registering of changes within their and the counter-part agent ontology.

The most challenging research domain has been the match quality evaluation. Measures like *precision* (the proportion of retrieved and relevant mappings to all the mappings retrieved) and *recall* (the proportion of relevant mappings that are retrieved, out of all relevant mappings available) [34, 39], have been borrowed from information retrieval domain. These metrics have been customized to quantify the quality of schema matching but still require a lot of work.

Table 3.3: Dimensions of Schema Matching - Application Domains

Z - Application Domains	**Type**
Data Warehousing	static
Message Translation	static
E-Commerce	static
Catalogue Integration	static/dynamic
Web Services Discovery and Integration	dynamic
Agents Communication	dynamic
Enterprise Information Integration	static/dynamic
Data Mashups	static/dynamic
Schema based P2P Database Systems	dynamic
Federated Systems	static/dynamic
Business Processes Integration	static
Query Answering (Web/Distributed Systems)	static/dynamic
Ontology Management	static/dynamic

The **application domains** for schema matching research can have a long list. Some prominent and latest fields are enumerated in table 3.3. The application domains can be categorized with reference to the time line and the data interoperability static or dynamic aspect. Late 80s and early 90s have been dominated by the static nature of matching. For example, in applications like Data Warehousing, Message Translation, E-commerce [86], the source schemas have been created and their matching and integration is one time fixed process. Whereas the applications of late 90s and current era, have a much dynamic nature propelled by the internet and its changing technologies. The concepts like Web Services, P2P Databases, Large Scale Querying [93], demand techniques which can support the independence and changing nature of contributing sources. A detail review of the current trends and related applications is given in section 3.3.

3.2.2 Large Scale Schema Matching

We take our motivation from the current trends of large scale dynamic aspect of schema matching. Large scale schema matching can be categorized into two types of problems depending upon the input, (i) two large size schemas (with thousands of nodes). For example bio-genetic taxonomies [24], (ii) a large set of schemas (with hundreds of schemas and thousands of nodes). For example hundreds of web interface forms (schemas) related to books domain [50, 102].

The schema matching tools available today can be used for applications which require matching of two large schemas. [24] and [78] demonstrate that with some modification to the available tools/infrastructures, the required goals can be achieved. Research in [24] breaks down the bio-genetic taxonomies into fragments and apply their matching tool COMA++ [3] on pairs of these fragments to find similarities between the two taxonomies. Whereas, work in [78] uses three levels of matching; using CUPID [66] for lexical analysis of nodes using external oracles, then applying Similarity Flooding [75], fix point computation algorithm based on the idea of neighborhood affinity, and in last phase the hierarchical matching finds similar descendants. The ideas work well in case of two schemas but when the scenario has large number of schemas, the formalization, techniques and algorithms for the problem change.

For us the motivating scenario lies in the integration of large number of schemas with automated matching aspect. Today this problem is specifically encountered in applications like schema based P2P database systems, query answering over the web, web services discovery/integration and data mashups in enterprise information integration. The problem has been researched using holistic matching or incremental pair-wise matching and integration algorithms, using recursive [75], clustering [76, 94, 102] and mining [50, 89] techniques. The automation factor is a must to solve this problem. Since large number of schema matching can not be handled semi-automatically, the notion of approximate semantic matching rather than exact match, with performance has been advocated [50, 102].

3.3 New Application Domains for Data Interoperability

Schema matching research has its roots in schema integration applications in distributed database systems. The task is to produce a global schema from independently constructed schemas. The requirements for such an integration have been

presented in [5, 95]. The research highlights the issues in schema integration of relational schemas, the integrity of integrated schema and different possible techniques to integrate schemas (binary or n-ary). Data Warehousing, Message Translation [86], E-commerce, B2B, B2C [93] applications are examples of implementation of this research.

Today, from the artificial intelligence view point the research in this domain revolves around *ontologies*. Ontology is a way to describe data elements along with inter-element relationship rules, based upon object oriented techniques but coded in a semi-structured way. In the last couple of years domain specific ontologies have been incorporated in the data integration processes, demonstrating acceptable results [35]. But the core problems faced in the changing world for communication and integration are the same, whether it is ontologies or schemas [47, 49] .

The latest trends in applications development requiring data interoperability can be explicitly attributed to the technologies harnessing the web. For example ontologies alignment [35], integration of XML data on the web [76] etc. In the subsequent subsections we give the current application domains motivating our work on schema matching.

3.3.1 Web Services Discovery and Integration

Initial concept of web was to share scientific research, followed by web sites for advertisement of products and services. Next the business community used it to do transactions with their customers, followed by secure business transactions between two e-business ventures, called B2B systems. This gave rise to the web service concept i.e., set of functions which can be invoked by other programs over the web. So, to achieve a certain goal, the user/program has to first discover the services, perform some matching to select the appropriate services, do some planning for execution of the services to get to the subgoals and finally combine the subgoals [54] to achieve the main goal. One approach to search for web services is to access a UDDI (Universal Description, Discovery, and Integration - standard for centralized service repositories) Business Registry (UBR) as the search point. Web service providers register their services with the UBRs for subsequent usage by others. Another approach is to use web search engines which restrict their search to WSDL (Web Service Description Language) files only [4]. WDSL is an XML based language standard for describing a web service. The need for matching and merging is quite evident, as web services have to be searched against user goal requirements, compared and integrated for subgoals achievement.

3.3.2 Data Mashups in Enterprise Information Integration

Data Mashups is the most recent buzz word in the Enterprise Information Integration (EII) domain. Its definition can be: making new knowledge by joining available information. Web mashups are emerging at a rapid pace. *Programmable.com* provides a list of such mashups. A typical web mashup joins information from related web sites. For example a mashup website about cars can get quotes about a certain car from quotes websites, pictures and reviews from cars forums along with video footage from some social network like *youtube.com*. Thus the information resources can range from a simple database table to complex multimedia presentation i.e., the search can be on any structured or unstructured data.

Thus the core concept in mashups is to extract some new necessary knowledge from all these sources existing in different formats. This is a new challenging issue in information extraction and integration. The research aim is to provide light and fast protocols which can work through different meta models and types of documents [49]. At the enterprise level, the mashup idea helps in building quick situational applications, for some transient need in the enterprise, complementing the more robust and scalable integration technologies that the enterprises invest in.

An example of enterprise mashup implementation is done at IBM as Information Mashup Fabric (MAFIA) [56]. In MAFIA the data input are complimented with those normally not covered by traditional EII systems, e.g., emails, presentations, multimedia data etc. In the coming years, mashups will open up a new enterprise application market, providing business users and IT departments with a quick and inexpensive approach to develop and implement applications, requiring matching and joining data in diverse formats.

3.3.3 Schema based P2P Database Systems

One of the latest emerging research field in databases over the web is *P2P Databases* [48]. There have been numerous successful P2P systems delivered in the last couple of years. Traditionally, the P2P systems have been simple file sharing systems which can self tune, depending upon the arrival and departure of contributing peers. Industrial-strength file sharing P2P systems, like Kazaa and bitTorrent, allow the peer autonomy of participation but they still restrict the design autonomy of how to describe the data. Secondly, sharing of data objects described by one P2P system are not available in another P2P setup. Today, the P2P technology has transformed into sharing of any kind of data, whether it is semi structured XML data or continuous multimedia streaming [73]. The next generation of data sources are going to be

totally independent of each other, i.e., they will have the design autonomy, utilizing their own terminologies for their data structuring, with capabilities to interact with others. For querying these data sources some matching method will be required to broker between their structures, giving rise to the new generation of application research of schema based P2P data sharing systems [63].

3.3.4 Querying over the Web

Query processing has two intrinsic problems; understanding the query and then finding the results for it. The web contains vast heterogeneous collections of structured, semi-structured and unstructured data, posing a big challenge for searching over it. Deep Web [50] scenario highlights this aspect. Firstly, the heterogeneity problem allows the same domain to be modeled using different schemas. As we have discussed in the example for our motivation. Secondly, it is very difficult to define the boundary of a domain. For example, traveling and lodging are inter-linked for tourist information web sites. Continuous addition of new content further complicates the problem for searching and integrating the results.

3.3.5 Online Communities

People have been using online spaces to communicate, since the beginning of the internet. Today, with the available resources for the web, these communities have mushroomed to an unprecedented level. These virtual connections of people are also called social networks. Every community has a purpose or goal with a target audience. For example, videos or photos sharing communities or simple forums regarding a specific subject. To be more business oriented, distributed work groups within companies and between companies use online community to build their team, keep in touch and even work on projects together. Sometimes, one can find more exact answers to queries from specific online community rather than from search engine.

Whatever the reason for the community, it needs a structure to support the underlying collaborative data. The data sources can be as diverse as discussed in mashups. In such virtual communities there is no central authority to monitor the structure and the performance. Users can join and leave, contribute or simply use the resources like P2P systems. In such a scenario, the matching of data resources is an extreme problem in schema matching domain. The problem complexity is further elevated if we consider an inter community communication. With the semantic web around the corner, this domain requires lots of attention.

There are very few studies in this area. In [72], authors show a question answer based technique to solve the match problem in online communities. The method automatically generates questions for element names which are not possible to be compared. Choices are presented to several users and then the results are heuristically evaluated to assess the correct match.

3.3.6 Agents Communication

Agents Communication can be considered as a dialogue between two intelligent entities. Each agent works out its actions according to its own intelligence or ontology. When two independent agents come in contact for the first time, they need some protocol to translate the message of one agent into the ontology of the other agent [93]. For subsequent encounters the agents may utilize the mappings discovered and stored within them. To answer the query of its user, an agent may have to interact with a number of other agents, compare and integrate their responses, just like web services. Only agents have inbuilt mechanisms to learn and counter the changes around them. P2P Ontology Integration [10] proposes a framework for agents communication in a P2P network. Its main feature is that, it efficiently tries to map dynamically only the parts of ontologies, which are required for the communication.

The above set of application domains have one thing in common, they encounter dynamic information requirements, changing over time and process web scale data. It is very difficult to achieve desired performance oriented goals with research revolving around semi-automatic schema matching and integration approach. The scenarios require an automatic intelligent and self-tuning solution.

3.4 Schema Matching Techniques

This section gives an overview of the basic techniques used in the schema matching and integration research. Schema comprises of some basic entities called elements. The composition of elements within the schema follows rules outlined by a data model. To date, a number of algorithms have been devised and implemented for finding correspondences between schemas. These algorithms have been dependent on techniques of string matching, linguistic similarities or constraints likeliness at element level or higher schema structure level. Graph algorithms utilised in schema matching are special form of constraints matching [93] for managing structural similarity. In some cases, these algorithms are further supported by data instances of

schemas.

3.4.1 Element Level

Schema matching is a complex problem, which starts by discovering similarities between individual schema elements. Every element, disregarding the level of granularity, is considered alone for a match. The techniques used, basically rely on the element's name and associated description, using basic **string matching** approaches adapted from the information retrieval domain [30]. These approaches include string prefix, suffix comparisons, soundex similarities and more sophisticated algorithms based on string distance. There is a large list of these algorithms with various variations researched over time. The mainly talked about approaches are the n-gram and the edit distance[1]. For example Google use n-gram for statistical machine translation, speech recognition, spelling correction, information extraction and other applications.

Linguistic techniques are based on the tokenisation, lemmatisation and elimination. The idea is to extract basic sense of the word used in the string. And then find its contextual meaning [13, 30] i.e., meaning extraction according to the elements around it. These techniques have been adopted from linguistic morphological analysis domain. The algorithms are further enriched to provide synonym, hypernym, hyponym similarities by using external oracles, dictionaries, thesauri like WordNet [41], domain specific ontologies or upper level ontologies [79].

Constraints similarity is data model dependent. One of the basic constraints found in almost every model is the element type e.g. integer, string etc. Different data models have their own lists of constraints. Relational model has primary key constraint to bind different attributes data in a tuple or foreign key constraint to relate to table elements. Similarly, is-a and has-a relationship constraints in object oriented model and parent-child relationship in hierarchical structure of XML data model. These relationship constraints help in extracting the relative contextual concept of an element.

3.4.2 Structure Level

Structure level matching is referred as matching a combination of elements from one schema to another schema [86]. The algorithms developed are based on graph

[1]Listing with detail available at http://www.dcs.shef.ac.uk/∼sam/stringmetrics.html

matching research. It can also utilize external oracles like known patterns [33], ontologies [27] or corpus of structures [65] to recognize the similarity. It also helps in solving n:m complex match problem.

Today, almost every schema matching implementation uses some form of **graph structures** for internal representation of schemas. Graph matching is a combinatorial problem with exponential complexity. Researchers use directed acyclic graphs or trees to represent schemas, ontologies or taxonomies, to reduce the complexity aspect of the problem. In generic schema matching tools (which can take as input different data model schemas) the graph structures are flexible enough to support the possible input schema elements and perform mapping. Nearly all schema match research projects based on graphs, use the notion of neighborhood affinity to compute the similarity match value for individual elements. This aspect has been presented in Similarity Flooding algorithm [74].

In large scale scenarios, structure level matching techniques help in enhancing the performance of the match implementations, by using neighborhood search algorithms [32]. In literature holistic [50] or level-wise algorithms (children-parent relationships) [66, 24] have been used to determine the correspondences among two schemas.

Another variation of structure level matching is based on taxonomy of ontologies. For example *bounded path matching* [32] takes two paths with links between classes, defined by the hierarchical relations, compare terms and their positions along these paths, and identify similar terms. *Super(sub)-concepts rules* oriented match follows the idea that if super-concepts are the same, the actual concepts are similar to each other. Another related interesting measure called *upward cotopy distance* [35] measures the ratio of common super classes to find similarity of classes of two taxonomies.

Structure level matching also follows model-based techniques. The graph (tree) matching problem is decomposed into a set of node matching problems. Each node matching problem is translated into a propositional formula, namely pairs of nodes with possible relations between them. And finally the propositional formula is checked for validity. Research in [43] demonstrates the effectiveness of this technique but with worst time performance, when compared to other available tools.

3.4.3 Use of Data Instances and Machine Learning

Data instance in schema matching is used in two ways. First, if the schema information is very limited or not available, instance data is used to create a representation of the data [11]. For example from any XML document, a basic tree hierarchy of

elements can be extracted. Even, if the schema is available, data instances can augment the schema matching by giving more insight about the schema element semantics [52]. For example city names encountered in data instances (found in a general list of city names) can help infer that the field is a component of address field.

In second case data instances are used in schema matching for training machine learning algorithms. In [27], XML schema inner nodes are matched by comparing concatenated values of their corresponding leave nodes using learning techniques, e.g., address is a composition of street, zip and city. In another research [22], n:m mapping expressions are predicted, involving arithmetic functions, like totalprice is equal to price+(price*taxrate). First, it uses an external global domain ontology to map the elements and then find the function by employing the data instances with a set of arithmetic and string concatenation rules .

The drawbacks in use of data instances can be either the bulk of data to be analysed, thus down-grading the performance or the verification of the quality and granularity of data instance, which may require some cleansing technique [59]. In the dynamic environment where the load of schemas itself is quite large, data instance approach is difficult to implement because of its drawbacks.

3.5 Match Strategies

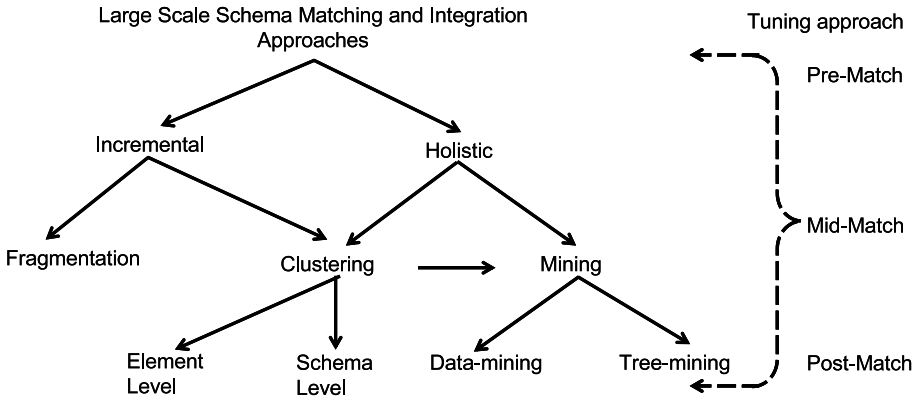

Figure 3.4: Taxonomy for Large Scale Schema Matching and Integration Strategies

Different schema match research projects have shown that single match algorithm is not enough to have a quality match. It is necessary to employ a range of algorithms, applied in a sequence or parallel, optimized for the application domain. Researchers have followed different strategies depending on application domain or researcher's objectives. The strategy is basically governed by the input and output requirements

of the match tool.

Input aspect of the tool outlines the information about the entities, available for matching. For example *schema-based vs data instance-based* or the tool is for some *explicit input domain* or not. The output requirements depend upon the research domain, in which the tool is to be utilized. The output can also dictate the tool to be *manual, semi-automatic* or *automatic*. For example, web services require an automatic environment and comparison of two large bio-genetic ontologies can be worked out with a semi-automatic tool, where possible matches are presented to the user to select the appropriate one as the mapping.

The execution part is responsible for rest of the categorisation of schema matching tools. Namely, *internal vs external* [86], *syntactic* vs *semantic* [93] and *Hybrid* [66, 43] vs *composite* [24] approaches. Some latest developments in matching approaches are being guided by the large scale scenarios like P2P data networks, semantic web, query over the web and semantic grid services. These large scale scenarios are being dealt with using techniques which can retrieve good match results directly or enhance [61, 77] the already existing results automatically. In some work, performance with approximate mapping is being preferred over exact mapping [50, 89].

In next sub-section, we give an account of the strategies adopted by the researchers or which can be exploited for large scale schema matching. Figure 3.4 shows a classification of these strategies, with inter-strategy relationships.

3.5.1 Schema Fragmentation Approach

In the domain of semi-structured data, more and more schemas are being defined in XML, a standard language adopted by W3C. It is being widely used in E-business solutions and other data sharing applications over the web. Over time, the emergence of distributed schemas and namespaces concepts has introduced more complexity to the matching problem.

Research work in [24] demonstrates, how these emergent problems can be tackled. The authors propose the idea of fragmentation of schemas for matching purposes. The approach, first creates a single complete schema , including the instances for the distributed elements or namespaces used in the schema. In second step the large schema instance is broken down into logical fragments which are basically manageable small tree structures. The tool COMA++ [3] is used to compare each fragment from source schema to each fragment of target schema for correspondences, with the help of GUI and human input. The approach decomposes a large schema matching problem into several smaller ones and reuses previous match results at the

level of schema fragment. The authors have reported satisfactory results.

In [53], the authors apply the fragmentation (partitioning) approach on large class hierarchies extracted from ontologies. Each partition is called a block with an anchor class. Matches for anchor classes are pre-detected, thus elements of blocks with similar anchors are further matched in the system.

3.5.2 Clustering Approach

Clustering refers to the grouping of items (items can be schemas or elements of schemas) into clusters such that items in one cluster are more similar to one another (high affinity) and those in separate clusters are less similar to one another (low affinity). The level of similarity can vary from application or technique which is using clustering approach. Since the schema matching problem is a combinatorial problem with an exponential complexity, clustering works as an intermediate technique and improves the efficiency of the large scale schema matching. In schema matching and integration, clustering can be considered at element level or schema level.

Element Level clustering can be applied on a single schema or holistically on the given set of schemas. The authors of [94] give a generic approach using the element level clustering method to detect element clusters in schema repository which are probably similar to a given personal source schema. Personal schema is then fully compared to detected list of clusters. So, rather comparing and applying all match algorithms on all schema elements in the repository, only a subset of elements are considered.

In our work [89](chapter 5), element clustering is applied at the holistic level of schemas. The work is directed toward large scale schema integration. Initially, a set of clusters is created, in which each cluster have linguistically similar label elements. Intuitively, the nodes having similar labels are also clustered together. The largest size schema in the input schemas is considered as initial mediated schema. Each input schema is compared to the mediated schema. The source element is only compared to the elements found in its cluster belonging to the mediated schema.

Schema Level clustering is an extended version of element level clustering. The approach clusters together schemas which show some level of elements' similarity among them. In [60], the authors demonstrate a recursive algorithm which finds similar elements in XML DTDs and creates their clusters. In second step, it performs the integration on each DTD cluster. The process goes on until one global DTD has been created.

A very comprehensive review on XML schema clustering techniques is given in [20].

3.5.3 Data Mining Approach

Data Mining is the technique for finding similar patterns in large data sets. Very recently, it has been used as schema matching method. Work in [50, 96] highlights this method for matching and integrating deep web schema interfaces. He et al. in [50] use a positive correlational algorithm based on heuristics of schema attributes. [96] applies negative correlational method to match and integrate schemas. Correlational mining refers to detection of elements existing together, along with their sequences, in a schema; and statistical evaluation of the frequencies of such existences.

Tree mining approach is a variation of data mining, in which data is considered to posses a hierarchical structure. It shows more affinity to XML schemas, which are intrinsically tree structures. Our approach, explained in chapter 5, demonstrates a method which combines the element clustering and a tree mining method. The work provides a time performance oriented solution for integrating large set of schema trees, resulting in an integrated schema along with mappings from source to the mediated schema.

3.5.4 Strategies for Enhancing Match Results

There has been a lot of work on schema matching but proof of exact results in the semantic world has been hard to achieve. In most of the research the quality of results has been said to be approximate [86, 80, 93]. As a result of these observations new avenues of research opened up for finding ways to achieve the maximum correctness in schema matching. Following are the approaches under active research.

Pre-Match Strategies: Pre-match methods typically deal with the matching tool's execution strategies, called *tuning match strategies*. These approaches try to enhance the quality of mappings of current schema matching tools, which have the ability to rearrange the hybrid or composite execution of their match algorithms. Defining external oracles, the criteria for their use and adjustment of parametric values, like thresholds, for different algorithms is also part of pre-match. The work in [61] provides a framework capitalizing on instance based machine learning. The authors describe the use of synthetic data sets to optimize a matching tool, for good match results when applied to a similar real scenario. The tuning module execution is totally separate from the actual tool working.

Post-Match Strategies: These strategies are concerned with improving the already obtained results from a schema matching tool. OMEN [77] Ontology Mapping En-

hancer, provides a probabilistic framework to improve the existing ontology mapping tools using a bayesian network. Is uses pre-defined meta-rules which are related to the ontology structure and the meanings of relations in the ontologies. It works on the probability that if one knows a mapping between two concepts from the source ontologies (i.e., they match), one can use the mapping to infer mappings between related concepts i.e., match nodes that are neighbors of already matched nodes in the two ontologies.

Manakanatas et al.[70] work is a post-match phase prototype application. It has an interface to detect the best map, from the set of mappings for a source schema element produced by COMA++. It uses WordNet as the linguistic oracle to filter the COMA++ results, in case there are more than one possible mappings for a source element to target schema. This minimises the human intervention.

One of the latest results in detecting the best match results, according to the user preferences, using fuzzy logic has been demonstrated in [46]. The work also enhances COMA++ results for deriving best semantic mappings. The research proposes to apply fuzzy sets theory utilizing pre-defined user preferences.

3.5.5 GUI aspect

User perception is getting more importance in the schema matching tools in the form of investments in the *graphical user interface* development for the generic schema matching tools. Current schema matching tools interfaces only support subject domain experts with good computer science background. And schema matching tools in large scale scenarios still lack the initiatives in user interface development. However, with matching becoming need of today, in the ever expanding data integration domain, new user centric graphical environments are emerging to support the matching tasks [100].

These environments have augmented the match task in *pre-match, amid-match and post-match phases*. Pre-match phase interface provide the facility to define a domain or application specific strategy, to align the different schema matching algorithms. It can include manual configuration of various parameters of the match algorithms selection or specification of auxiliary information like synonyms, abbreviations and other domain specific constraints [3].

The post-match phase uses different measures to select the best correspondence, for an element from a set of possible matches which show the semantic equivalence aspect for that element [3, 8, 52]. Tools like CLIO [52], COMA++ [3] and Microsoft BizTalk Mapper [8] generate the possible mappings along with degree of match. And then graphically allow the user to select the mappings according to her expertise

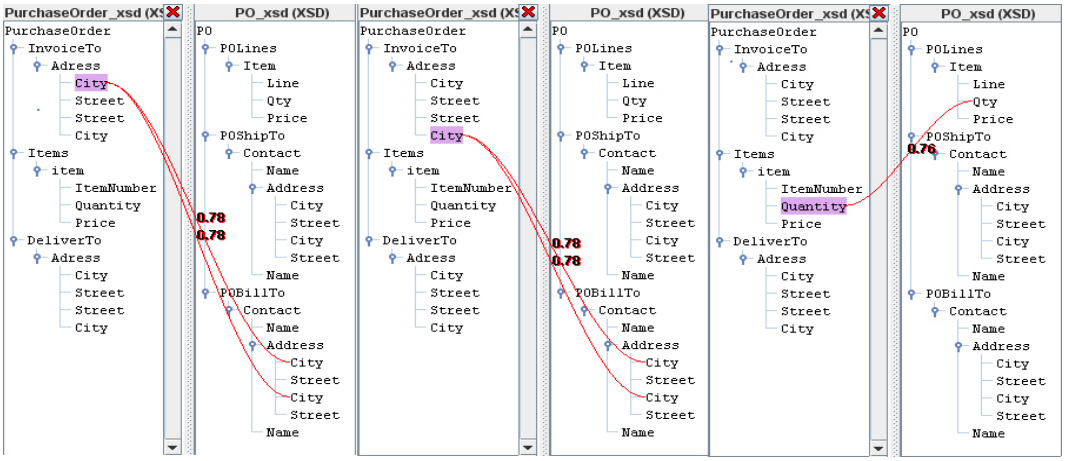

Figure 3.5: Three cases for selecting the best match: COMA++

(figure 3.5).

The amid-match phase interface interactively solves the matching problem, with the user help. Schema matching environment SCIA [100] (explained in section 3.5.1) provides a much detail interface. It focuses on minimum user intervention based on some pre-defined rules regarding the contextual matching of elements. The match process executes, matching schemas trees in a top down way. If the system encounters problem in matching some non-leaf node, it pauses and demands the user to provide a match solution.

3.5.6 Top-k Methods

Top-k mappings method, semi-automatically, tries to find not the best but k best possible matches from which user can select the most appropriate. Thus, intuitively increasing the recall measure of the quality of mappings. There exists two variations (i) *element level* and (ii) *schema level* top-k mappings. In former case the matches are presented for each element. Whereas in latter, top-k possible sets of mappings for whole schema are considered.

In literature, there are very few works regarding schema level top-k mappings. Several schema matching tools analyse the target search space in an iterative manner, which can be considered as top-K approach. Most of these tools, find element level top-k matches. For example, the CLIO [52] tool calculates the best matches and the user has to select or reject the matches. A rejection results in re-evaluation for next best possible match for that element. Thus producing a highly user dependent iterative system. LSD [26] also works in the similar fashion; accepting the rejection as a constraint for its learning process. In contrast, COMA++ [24], presents the user with the best matches with match confidence higher than the defined threshold.

There is no fix value for k in this case. The user can select one of the proposed matches or reject all. Another approach presented in QOM [32], also works in the similar fashion; reutilising the results from previous iteration and proposing a new set of k matches. It demonstrates a more robust, almost automatic approach with good quality and time performance results.

[38] presents an automatic approach which finds top-k possible sets of mappings between two schemas. Next, the similarity between the sets is analysed heuristically, resulting in a set of mappings which are most frequent in all the sets. The author gives very solid results and arguments to support the high recall value for this set.

Another variation of the technique is discussed in [90]. The authors demonstrate an automatic technique for generating top-k mediated schemas along with mappings from input schemas to the integrated schemas. First, the algorithm generates possible clusters of similar elements among the input schemas and then produces a set of probabilistic mediated schemas with probability values i.e., top probable schemas. Further, the probabilistic mappings are computed from input schemas to the probabilistic mediated schemas. Queries from users are applied on the probabilistic mediated schemas and data results are statistically evaluated for same query. Similarity in results, provides a way to verify the integrity of probabilistic mediated schemas and helps in constructing the deterministic mediated schema with mappings.

A somewhat similar approach as above is given in [19]. Authors demonstrate a method to generate all possible integrated schemas, without duplicates, from a set of input schemas, along with mappings. These schemas are merged, based upon user defined constraints in an interactive manner to generate the final integrated schema.

3.5.7 Discussion

In the preceding sub-sections, we have discussed some recent strategies to enhance the quality of results of schema matching and integration. These techniques supplement the already existing basic schema matching and integration algorithms, and also highlight the fact that structural comparison of schemas is an essential part of schema matching process. The semantic aspects or concepts hidden in the schemas can be extracted with the help of algorithms exploiting the structures of schemas or taxonomies of ontologies. These algorithms search for contextual meaning of each node within the graph(tree) structure representing the schema/ontology.

Another aspect, quite evident in schema matching research is the use of GUI and interactive user input at pre-match, post-match and during the match process. The probabilistic, uncertainty and fuzzy logic based methods are also being exploited at data and schema level, to come up with good map results for data interoperability.

Use of clustering and data mining approaches are becoming more frequent to tackle the large scale scenario.

The *quality evaluation* of schema mapping is also an open research problem. It can be divided into two parts, *Correctness* and *Completeness*. Correctness follows the idea that the mappings discovered are correct, and completeness means every possible mapping has been discovered. The current measures utilised to evaluate the quality of a match tool, have been derived from the information retrieval domain. Specifically the *precision* measure and the *recall* measure are most widely used to verify the quality [23]. Some variances of recall and precision are given as *F-measure*, the weighted harmonic mean of precision and recall measures, and *Fall-out* which is the proportion of irrelevant mappings that are retrieved, out of the all irrelevant mappings available. A theoretical and empirical evaluation of schema matching measures is explained in [34, 40].

3.6 Overview of Large Scale Schema Matching Tools

The previous surveys [86, 93, 103] incorporate solutions from schema level (metadata), as well as instance level (data) research, including both database and artificial intelligence domains. Most of the methods discussed in these surveys compare two schemas and work out quality matching for the elements from source schema to target schema. Some of the tools also suggest the merging process of the schemas, based on the mappings found in match step. In this section, we review the effectiveness of schema matching tools with respect to large scale scenario.

3.6.1 Tools: Matching Two Large Schemas

COMA++[3] is a generic, composite matcher with very effective match results. It can process the relational, XML, RDF schemas as well as OWL ontologies. Internally it converts the input schemas as graphs for structural matching and stores all the information in MYSQL as relational data. At present it uses 17 element/structure level matchers which can be selected and sequenced according to user's requirements. For linguistic matching it utilizes user defined synonym and abbreviation tables, along with n-gram name matchers. Structural matching is based on similar path and child/parent similarities.

Similarity of pairs of elements is calculated into a similarity matrix. It has a very comprehensive graphical user interface for candidate match selection and merging. For each source element, elements with similarity higher than the threshold are dis-

played to the user for final selection. The COMA++ supports a number of other features like merging, saving and aggregating match results of two schemas for reuse. An approach described in [24], uses COMA++ for matching large taxonomies by fragmenting them. The source and target schemas are broken down into fragments. Each source schema fragment is compared to the target schema fragments one by one for a possible match. Then, best fragment matches are integrated to perform schema level matching. Thus, this approach provides a mechanism for large scale matching and merging of two schemas.

PROTOPLASM [9] target is to provide a flexible and a customizable infrastructure for combining different match algorithms. Currently CUPID [66] implementation and Similarity Flooding (SF) [74] algorithms are being used as basic matchers. A graphical interface for it has been proposed and demonstrated by the name of BizTalk Mapper [8]. It is based on the human computer interaction research presented in [42] and is very heavily dependent on Microsoft technologies. PROTOPLASM supports numerous operators for computing, aggregating, and filtering similarity matrices. By using a script language, it provides the flexibly for defining and customizing the work flow of the match operators. SQL and XML schemas, converted into graphs internally, have been successfully matched.

Mork and Bernstein [78] present a case study of matching two large ontologies of human anatomy, using PROTOPLASM infrastructure. They use an extended version of hierarchical algorithm, which goes one step further than COMA++. The similarity of descendants is used to evaluate ancestor similarity, to child-grandparent level. The authors argue that the hierarchical approach produced disappointing results because of differences in context. They report that a lot of customization was required to get satisfactory results.

CLIO [52] has been developed at IBM. It is a complete schema mapping and management system. It has a comprehensive GUI and provides matching for XML and SQL schemas (Object Relational databases converted into relational with the help of a wrapper function). It uses a hybrid approach, combining approximate string matcher for element names and Naive Bayes learning algorithm for exploiting instance data. It also facilitates producing transformation queries (SQL, XQuery, or XSLT) from source to target schemas, depending upon the computed mappings. Its interface gives the user the facility to augment the schema semantics or the data instance (to support users expertise) in the pre-match phase and selection of best among the candidate matches in the post-match phase.

Another project, **ToMAS** [99], has added a new module to CLIO. It arms CLIO with the capability to handle the temporal changes in already mapped schemas and produce the required changes in the existing mappings. The work also presents a simple and powerful model for representing schema changes

SCIA [100] is a semi-automatic schema mapping system for data integration. It creates executable mappings in the form of views between two schemas, similar to CLIO. It provides an automatic matching mechanism for simple element level matching. In parallel, it finds points in the schemas, where user input is necessary. These points are computed where there exists ambiguous contextual information of a pair of matching elements. The authors research is based on the argument that human perception works well to select a better mapping from a given set of possible matches. But humans are not good and fast enough to identify mismatched portions of the schemas and matches missed by the tool. The system handles schemas as tree structures and tries to find context based matches. During the matching process, it interactively asks specific questions to resolve these problems. For example, SCIA asks the user how to proceed, if no match is found for a non-leaf node, with a significantly large subtree rooted at that node.

QOM (Quick Ontology Matching) [32] is a semi-automatic ontology (RDF based) mapping tool. It uses heuristics to classify candidate mappings as promising or less promising. It uses multiple iterations, where in each iteration the number of possible candidate mappings is reduced. It employs label string similarity (sorted label list) in the first iteration, and afterward, it focuses on mapping change propagation. The structural algorithm follows top down (level-wise) element similarity, which reduces time complexity. In the second iteration, depth-first search is used to select the appropriate mappings from among the candidate mappings. QOM has been incorporated into a complete schema matching tool called FOAM (Framework for Ontology Alignment and Mapping) [31].

GLUE [27] is the extended version of *LSD (Learning Source Descriptions)* [26], which finds ontology/ taxonomy mapping using machine learning techniques. The system is input with set of data instances along with the source and target taxonomies. Glue classifies and associates the classes of instances from source to target taxonomies and vice versa. It uses a composite approach, as in LSD, but does not utilize global schema (as in LSD). LSD uses composite approach to combine different matchers (a meta-learner combines predictions of several machine learning based

matchers).

LSD has been further utilized in *Corpus-based Matching* [65], which creates a corpus of existing schemas and their matches. In this work, input schemas are first compared to schemas in the corpus before they are compared to each other. Another extension based on LSD is *IMAP* [22]. Here, the work utilizes LSD to find 1:1 and n:m mapping among relational schemas. It provides a new set of machine-learning based matchers for specific types of complex mappings expressions. For example, name is a concatenation of firstname and lastname. It also provides the information about the prediction criteria for a match or mismatch.

3.6.2 Tools: Matching and Integrating Large Set of Schemas

MOMIS [6] is a heterogeneous database mediator. One of its components ARTEMIS is the schema integration tool which employs schema matching to integrate multiple source schemas into a virtual global schema for mediation purposes. The tool operates on hybrid relational-OO model. It first calculates elements similarity based on name and data type, thus acquiring all possible target elements. Further, external dictionary WordNet is utilized to compute the synonym, hypernym relationship between elements. In next step, structural similarity of elements is computed as the fraction of the neighbor elements showing name similarity exceeding a threshold over all neighbor elements. For each pair of elements, the name and structural similarity are aggregated to a global similarity using a weighted sum. According to the global similarities, similar elements are clustered using a hierarchical clustering algorithm for supporting complex match determination.

Wise-Integrator [51] is a schema integration tool. It uses schema matching to find correspondences among web search forms so that they can be unified under an integrated interface. First, a local interface is selected and then incrementally each input form is compared against it. The attributes without a match candidate in the local interface, are added to it. Wise-Integrator employs several algorithms to compute attribute similarity. Namely exact and approximate string matching, along with dictionary lookup for semantic name similarity. It also utilises specific rules for compatibility of data types supported by value scales/units and default values. For each pair of elements, the similarities predicted by the single criteria are simply summed to obtain a global weight. Elements showing the highest global weight exceeding a threshold are considered matching. One of the elements from each matchings pair, is selected as the global attribute to be used in the integrated interface.

DCM framework (Dual Correlation Mining) [50] objective is similar to Wise-Integrator. It focuses on the problem of obtaining an integrated interface for a set of web search forms holistically. The authors observe that the aggregate vocabulary of schemas in a (restricted) domain, such as book, tends to converge at a small number of unique concepts, like author, subject, title, and ISBN; although different interfaces may use different names for the same concept. The research proposes a statistical approach, extracted from data mining domain, based on the assumptions: independence of elements, non-overlapping semantics, uniqueness within an interface, and the same semantics for the same names. The algorithm identifies and clusters synonym elements by analyzing the co-occurrence of elements in different interfaces.

PSM (Parallel Schema Matching) [96], is another implementation of holistic schema matching, for a given set of web query interface schemas. The objectives are similar to DCM algorithm, but PSM improves on DCM on two things; first DCM negative correlation computation between two elements to identify synonyms may give high score for rare elements but PSM does not. And secondly the time complexity of DCM is exponential with respect to the number of elements whereas for PSM it is polynomial. PSM, first holistically detects all the distinct elements in the input schemas, assuming synonym elements do not coexist in the same schema. In second phase, it generates pairs of candidate synonym elements. This pair generation is dependent on a threshold calculated by the number of cross-occurrences (if element1 is in schema1 and element2 is in schema2 or vice versa) in different pairs of schemas. The results of the experiments in this work show that it has the ability to find 1:1 and n:m matches quite efficiently.

ONTOBUILDER [87] is a generic multipurpose ontology tool, which can be used for authoring, and matching RDF based ontologies. Its interface also supports the process of matching web search forms for generating an integrated form. OntoBuilder generates dictionary of terms by extracting labels and field names from web forms, and then it recognizes unique relationships among terms, and utilizes them in its matching algorithms. The tool uses spacial attribute precedence based algorithm to calculate the semantics of each attribute in the form i.e., sequencing of concepts within the form.

PORSCHE (Performance Oriented Schema Matching) [89] presents a robust mapping method which creates a mediated schema tree from a large set of input XML

schemas (converted to trees) and defines mappings from the contributing schema to the mediated schema. The result is an almost automatic technique giving good performance with approximate semantic match quality. The method uses node ranks calculated by pre-order traversal. It combines tree mining with semantic label clustering which minimizes the target search space and improves performance, thus making the algorithm suitable for large scale data sharing. The technique adopts a holistic approach for similar elements clustering in the given set of schemas and then applies a binary ladder incremental [5] schema match and integrate technique to produce the mediated schema, along with mappings from source schemas to mediated schema.

Bellflower is a prototype implementation for work described in [94]. It shows how personal schema for querying, can be efficiently matched and mapped to a large repository of related XML schemas. The method identifies fragments within each schema of the repository, which will best match to the input personal schema, thus minimizing the target search space. Bellflower uses k-means data mining algorithm as the clustering algorithm. The authors also demonstrate that this work can be implemented as an intermediate phase within the framework of existing matching systems. The technique does produce a time efficient system but with some reduction in quality effectiveness.

3.6.3 Summarizing the Tools

In tables 3.4 and 3.5 we give a quick comparison of the above discussed schema matching tools and prototypes. The comparison in table 3.4 has been devised to give a general outlook of tools, highlighting the use of GUI, match cardinality supported by the tool, use of external oracles and related application domains. Table 3.5 gives a much deeper insight into the algorithms used by the tools with respect to input, output and execution aspects.

The analysis of the prototype tools for schema matching or ontology alignment domains shows that most of the techniques used are the same. For example, two most cited schema matching tools PROTOPLASM and COMA++ follow similar match characteristics and architecture, the only difference is that PROTOPLASM framework is hybrid in nature whereas COMA++ is composite, thus providing more flexibility. The tools adopt hybrid techniques for better and automatic approach. Structure level matching has been adopted by all, except for some web search interface schema integrators, since query form field attributes follow more of a sequence than hierarchical structure. For semantic comparison of element labels, external

Table 3.4: Schema Matching Tools and Prototypes Comparison - General

Tool	GUI	Approach	Card.	Ext Orc	Internal Rep	Research Domain
BELLFLOWER	No	Hybrid	1:1	-	Directed Graph	Schema Matching
CLIO	Yes	Hybrid	1:1	-	Rel. Model, Directed Graph	Schema Matching, Mapping Evolution
COMA++	Yes	Composite	1:1	Domain Syn, Abr Thesuri	Directed Graph	Schema Matching and Merging
DCM	No	Hybrid	n:m	-	-	Schema Integration
GLUE	No	Composite	n:m	-	Attribute based	Data Integration
MOMIS	Yes	Hybrid	n:m	Thesuri	Directed Graph	Schema Integration
ONTO BUILDER	Yes	Hybrid	1:1, 1:n	-	Graph	Create/Match Ontologies
PORSCHE	No	Hybrid	1:1,1:n	Domain Syn, Abr Thesuri	Tree	Schema Integration and Mediation
PROTOPLASM	Yes	Hybrid	1:1	Wordnet	Graph	Schema Matching
PSM	No	Hybrid	n:m	-	-	Schema Integration
QOM	No	Hybrid	1:1	Dom. Thesuri	Tree	Ontology Alignment
SCIA	Yes	Hybrid	n:m	Thesuri	Tree, Graph	Data Integration
WISE INTEGRATOR	Yes	Hybrid	1:1	General Thesuri	Attribute based	Web Search form Integration

oracle like WordNet dictionary or reference domain ontology is quite frequent. The notion of neighborhood likelihood for next possible match is followed by most of the matching tools e.g., PROTOPLASM, MOMIS, QOM, SCIA and GLUE. This feature is also intuitively used for search space optimization. Another characteristic for search space optimization in large scale scenario is clustering of elements/schemas, showing some similarity at the pre-processing level e.g., element name similarity based on edit distance or synonymous meaning, demonstrated in XClust [60] and QOM.

It appears that most prototypes aim to provide good quality matchings, with lack in time performance. Today, the application domains like the genomic or e-business, deal with large schemas. Therefore, the matching tool should also provide good performance and if possible automatic mapping generation. In the future, matching systems should try to find a trade off between quality and performance. A recent work in this domain has been proposed in [28], which uses decision tree concept based on machine learning algorithm.

3.7 Conclusion

In this chapter we have provided a broad overview of the current state of the art of schema matching, in the large scale schema integration and mediation for data in-

Table 3.5: Schema Matching Tools and Prototypes Comparison - Strategy based

Tool	**Input**	**Output**	**Match Algorithms (Level wise)**			
			Element			**Stucture/(Data Ins.)**
			Str.	*Ling.*	*Const.*	
BELLFLOWER	XSD	Schema Matches	Yes	-	-	K-means data mining
CLIO	SQL,XSD	Mappings (Query)	Yes	-	Yes	(Naive Byes Learner)
COMA++	XSD,XDR, RDF,OWL	Mappings, Merged Schema	Yes	Yes	Yes	Path: biased to leaf nodes
DCM	Web Query Interface	Mappings between all input schemas	Yes	-	Yes	Correlational Mining
GLUE	DTD,SQL, Taxonomy	Mappings, IMap functions	Yes	-	Yes	(Whirl/Bayesian Learners)
MOMIS	Rel,OO data model	Global View	Yes	Yes	Yes	Schema Clustering, Neighborhood Affinity
ONTO BUILDER	RDF	Mediated Ontology	Yes	Yes	-	Elements Sequencing
PORSCHE	XSD Instance	Mediated Schema	-	Yes	-	Elements Clust, Tree Mining
PROTOPLASM	XDR, SQL,RDF	Mappings	Yes	Yes	Yes	Path (Parent,Child,Grand Child), Iterative Fix Point Computation
PSM	Web Query Interface	Mappings between all input schemas	Yes	-	Yes	Correlational Mining
QOM	RDF(S)	Mappings	Yes	-	Yes	Neighborhood Affinity, Taxonomic Structures
SCIA	Rel,DTD, XSD,OWL	Mappings (Query)	Yes	Yes	Yes	Iterative Fix Point Computation, Path
WISE INTEGRATOR	Web Query Interface	Integrated Schema	Yes	Yes	Yes	Clustering

teroperability. We also tried to provide an insight on current emergent technologies driving the match research, like data mashups and P2P database networks. We have seen in this study that although two decades have passed but there are issues that still require to be investigated in schema matching and integration. This is because of the dynamic nature of today's application domains. Future prospective of schema matching is in the large scale, which is mainly related to schemas and ontologies in P2P, data grids, agents , web services and virtual social communities based networks. The schema matching research and tools study have provided us with the insights about the requirements for schema matching and integration application development. Based on these insights, next chapter explains the desiderata for the large scale schema matching and integration, before we move on to our approach in chapter 5.

Chapter 4

Requirements and Ingredients for Large Scale Schema Matching and Integration

Schema matching and integration relies on discovering correspondences between similar elements in a number of schemas. Several different types of schema matching [9, 24, 27, 43, 64, 66, 86, 93] have been studied, demonstrating their benefits in different scenarios. And in data integration schema matching is of central importance[5]. In this chapter we present the requirements and constraints in the large scale schema matching and integration scenarios. The content of this chapter has been chalked out after analysing the state of the art in schema matching.

4.1 Matching, Mapping and Integration

Most mapping tools match two schemas with human intervention [9, 27, 24, 43, 32, 64, 66]. The goals of research in this field are typically differentiated as matching, mapping or integration oriented. A *Matching* tool finds possible candidate correspondences from a source schema to a target schema. A *Mapping* is an expression which distinctly binds elements from a source schema to elements in the target schema, depending upon some function. *Integration* is the process of generating a schema which contains the concepts present in the source input schemas. Another objective, *mediation*, is mapping between each source schema and the integrated schema. The objective behind our work is to explore the ensemble of all these aspects in a large set of schema trees, using scalable syntactic and semantic matching and integration techniques. The target application area for our research is a large

scale scenario, like WSML[1] based web service discovery and composition, web based e-commerce catalogue mediation, schema based P2P database systems or web based querying of federated database systems.

4.2 Match Cardinalities

Match cardinality defines the participation ratio of elements in matches. It is driven by the level of schema granularity and automation allowed by a certain matching framework. For example, finest level of granularity in XML schema matching can be comparison of attributes of elements in the schema or in a relational schema the attributes within a table [86]. Considering the other constraints within the schema makes the match granularity more coarser and uncertain, e.g., a complex XML element is matched to a simple XML element. Pertaining to these considerations, match cardinality is categorized as follows:

4.2.1 Local Cardinality

The matches which are defined at element level between the elements of two schemas are basically called local matches. These can be:

- **Simple Match** is the most basic element level match type, given as
  - (i) 1:1 match - one element from source schema is matched to one element in the target schema
  - (ii) 1:n match - one element from source schema is matched to a combination of more than one elements in the target schema
  - (iii) n:1 match - a combination of more than one elements from source schema is matched to one element in the target schema
- **Complex Match** is structural level match, given as (iv) n:m match - a combination of more than one element from source schema is matched to a combination of more than one elements in the target schema (Example 4.1).

4.2.2 Global Cardinality

The overall matches between two schemas can also show another aspect, i.e., the number of times a certain element from a schema participates in the matches between the two schemas. If every element only participates in at most in one match,

[1] Web Service Modeling Language, http://www.w3.org/submission/WSML

we call it *simple global cardinality* and if any element shows its participation in more than one match, it is called *complex global cardinality*. The global cardinality is stated at schema level as 1:1, 1:n, n:1 and n:m.

Example 4.1 (Complex Match): A concept "totalPrice" in source schema is mapped to a combination of "price" and "tax" elements in target schema, with the mapping expression calculated as $(totalPrice)_{source} \leftrightarrow (price + (price * tax))_{target}$.•

Generally, in schema matching literature, one finds the discussion on local cardinality. However, the schema matching and integration research requires consideration on the global cardinality issue also. Allowing complex global cardinality can create problems in an automatic matching and integration process. However, a semi-automatic match tool for matching two schemas with complex global cardinality allowed, like COMA++ [3] or SCIA [100], augments the quality of match results.

4.3 Desiderata for Schema Mediation

Schema mediation can be defined as integration of a set of input schemas into a single global schema, with concepts mappings from the input schemas to the integrated schema, also called the mediated schema. There are numerous issues in the semantic integration of a large number of schemas. Beside mapping quality, the performance and integrity of the integrated schema are also very important. For example, the Semantic Web, by definition, offers a large-scale environment where individual service providers are independent. In such a situation the mappings can never be exact, rather they are approximate [27, 43, 50]. And with hundreds of services available, searching and selecting the required service needs to be fast and reliable enough to satisfy a user query. Following, is a brief discussion of the desiderata for schema integration and mediation.

4.3.1 Feasibility and Quality of Schema Mapping and Integration

The quality of mappings depends on the number and types of matching algorithms and their combination strategy, for instance their execution order. To produce high quality mappings, matching tools apply a range of match algorithms to every pair of source and target schema elements. The results of match algorithms are aggregated

to compute the possible candidate matches which are then scrutinised by users who select the best/most correct candidate as the mapping for the source schema element(s) to the target schema element(s). COMA++[24] and S-Match[43] follow this technique and create a matrix of comparisons of pairs of elements and then generate mappings. The final mapping selection can also be automated, depending upon some pre-defined criteria (match quality confidence), for example, choosing the candidate with the highest degree of similarity. QOM[32] and RONDO[75] use a variation of these techniques. The mapping algorithms used are applied to each pair of schemas of size amenable to human inspection. Large schemas, anything in excess of 50 elements, make human intervention very time consuming and error-prone. Therefore, automated selection of best mappings from the source to the target schema is a must in a large scale scenario.

The second aspect regarding the large scale scenario is the requirement for batch schema integration, where schemas may contain thousands of elements. Often, an integrated schema is to be created and mappings from source schemas to the integrated schema have to be produced, and this is to be done quickly and reliably. This requires both matching and integration. These tasks are not supported by most of the existing tools. For example COMA++, S-Match, QOM and RONDO do not provide them.

4.3.2 Schema Integration Approaches and Integrity Measures

Integrating a **batch of schemas** is a specific application of schema matching. Schema integration can be holistic, incremental or iterative (blend of incremental and holistic), as surveyed in [5]. The binary ladder (or balanced incremental integration) approach can be implemented as a script which automates the merging process, based on algorithms like QOM and SF. For example, in a binary ladder implementation, the result of merging the first two schemas is consecutively merged with subsequent schemas. [50, 96], on the other hand, discuss mining techniques and apply an n-ary integration approach (all schemas exploited holistically), to generate an integrated schema, along with approximate acceptable mappings. An iterative approach first finds clusters of schemas, based on some similarity criteria, and then performs integration at cluster level iteratively[60, 85, 94]. Although the iterative process is automatic and robust, the mediation aspect is very difficult to implement on top of those approaches.

The main purpose of schema integration in an information system is to hide the complexity of the underlying data sources and schemas from the user. The user accesses the integrated schema (mediated schema) to find a service or an answer

to a query. Batista and colleagues [71] explain the quality aspects of the integrity of a mediated schema in an information system. They highlight three quality criteria.

Schema completeness, computed as *completeConcepts/allConcepts* is the percentage of concepts modelled in the integrated schema that can be found in the source schema. Completeness is indicative of the potential quality of query results.

Minimality, calculated as *nonRedundantElements/allElements*, measures the compactness of the integrated schema and is normally reflected in query execution time.

Type consistency, measured as *consistentElements/allElements* spans all the input and mediated schemas. It is the extent to which the elements representing the same concept are represented using the same data types. It is a measur of congruence of input schemas.

4.3.3 Validity of Mappings

In the large scale scenario, it is very difficult to evaluate the quality of discovered mappings. The only possible way is to apply the developed approach on small enough scenario, whose results can be manually verified. The resulting mappings are evaluated in the light of the standard quality assurance measures. In literature [23, 34, 39], there are two main measures extracted from the information retrieval research domain:

- *Precision Measure*: The proportion of retrieved and relevant mappings to all the mappings retrieved
- *Recall Measure*: The proportion of relevant mappings that are retrieved, out of all relevant mappings available

Another measure, *F-measure* (similar to Fmeasure(α) in information retrieval) is a combination of precision and recall measures. It is given as

$$2 * \frac{(Precision * Recall)}{(Precision + Recall)}$$

Authors in [74] developed another measure , *Accuracy*, to estimate the effort required to identify the non-relevant retrieved mappings and adding the relevant mappings not retrieved. It is also known as *overall* [23] and given as

$$Recall * (2 - \frac{1}{Precision})$$

4.3.4 Time Performance

Time performance is an open issue in schema matching [86, 93], governed by schema size, the number of schemas to match and matching algorithms. The complexity of the matching task for a pair of schemas is typically proportional to the size of both schemas and the number of match algorithms employed, i.e. $O(n_1 x n_2 x a)$, where n_1 and n_2 are the counts of elements in the source and the target and a is the number of algorithms used [24, 43, 66]. Selection of the best matches from a set of possible matches increases the complexity of the problem. In schema integration and mediation, in addition to the parameters discussed above, as there are more than two schemas, we may consider instead the batch size (number of schemas) and the order in which the schemas are compared and integrated. The performance of the integration and mapping process can be improved by optimising the target search space for a source schema element. Minimising the search space by clustering will improve time performance.

4.4 Schema Mediation Tool GUI Features

In chapter 3 we gave a discussion on matching tools like SCIA [100], COMA++ [3], CLIO [52] and BizTalk Mapper [8]. These tools have graphical user interfaces (GUI) supporting the match process. Analysis of these tools helped us in chalking out the general features a tool must possess. The study also showed us that the GUI features should support the three possible phases of the matching process, as given below:

- **Pre-match phase** : Phase for pre-processing of the input schemas for matching and adjustment of different match confidence parameters.
- **Mid-match phase** : Interface for user intervention supporting and streamlining the on-going match process.
- **Post-match phase** : Phase supporting the refinement of the matches discovered by the tool's algorithms.

The above mentioned tools have been designed primarily for matching two schemas, pre-processed into tree structures. The trees are shown in two adjacent panels. COMA++ provided a composite approach for the use of algorithms, providing an interface for selecting/editing the match algorithms for optimal results. In all tools the match results are shown as lines flowing between elements of two schemas, supporting global cardinality, with each match line demonstrating the match confidence

value. The user has the option to select the best results, according to her expertize/knowledge. SCIA provided a unique mid-match feature: where the algorithms could not work out a contextual aspect of match for a certain element, the match process was halted and user prompted to solve the problem (with options, in the form of questions). The work showed that this helped in finding better subsequent matches. CLIO and SCIA also provided the facility to generate SQL/XQuery queries as mapping expressions.

These tools provide the basic GUI techniques required to match and merge two small schemas (with fewer than 100 elements). But in case of a large scale scenario for schema matching and integration, where the number of schemas can be more than two, or each schema size goes into thousands, we need to explore some better graphical representations without decreasing the overall performance. That requires optimizing the memory usage of the system.

Another point which surfaced during the study of schema matching, was the lack of a proper framework for benchmarking the tools. We evaluated a match algorithm based on expert matches. We feel that the GUI objects used in the matching tools could be customised to produce a simpler multi-user interface for marking the matches by the experts, in a collaborative manner. This could automate the benchmarking process for the tool for the large scale scenarios and the evaluation of performance in different domains.

4.5 Mining Techniques in Schema Matching

Mining for data or knowledge is generally viewed as the process of analysing large amounts of data and extracting relevant information. It revolves around finding data patterns that are frequent in the given large amount of data, which is similar to schema matching in that it tries to find similar concepts among a set of schemas. Data mining techniques are classified according to the structure of the data, for example, raw text data, relational data, graph data, sequential streaming data constrained by time, etc.

Data mining potential for large scale schema matching has been demonstrated in research [50, 96], as discussed in chapter 3. Bernstein et al. in [9] propose the application of data mining in their framework for schema matching. But work in [50, 96] demonstrates how schemas can be abstracted as transactions, since traditionally data mining is applied on transactional data. The authors apply positive and negative correlational mining of attributes in the schemas, demonstrating the power of data mining algorithms in finding complex matches in the deep web scenario.

Tree mining techniques extract similar sub tree patterns from a large set of trees and predict possible extensions of these patterns. In pattern-growing techniques, the pattern size starts from one and is incrementally augmented. There are different techniques [2, 104] which mine rooted, labelled, embedded or induced, ordered or unordered sub-trees. The basic function of tree mining is to find sub-tree patterns that are frequent in the given set of trees, which is similar to schema matching activity that tries to find similar concepts among a set of schemas abstracted as trees. Keeping this idea in mind, we researched a cross-disciplinary approach, i.e., schema matching and integration using tree mining approach with high quality matches and good time performance.

4.6 Conclusion

In this chapter we have summarized the desiderata for large scale schema matching and integration. We explicitly differentiated between the matching, mapping and integration processes. The issue of match cardinality in schema integration has been outlined. Other than the quality of matches and integrity of mediated schema, the importance of the time performance aspect has been also presented. The possible role of mining techniques in large scale schema integration with reference to data mining and tree mining has been covered in this chapter. Further, we gave some basic graphical user interface requirements in schema matching tool development.

In next chapter, we present our approach for schema matching and integration which uses a hybrid technique to match and integrate schemas and create mappings from source schemas to the mediated schema. To enhance the time performance and lower the cost of data integration, we try to remove the need for human intervention.

Chapter 5

PORSCHE: Simple Schema Matching and Integration

Semantic matching of schemas in heterogeneous data sharing systems is time consuming and error prone. Existing mapping tools employ semi-automatic techniques for mapping two schemas at a time. In a large-scale scenario, where data sharing involves a large number of data sources, such techniques are not suitable. In this chapter, we present a new robust automatic method which discovers semantic schema matches in a large set of XML schemas, incrementally creates an integrated schema encompassing all schema trees, and defines mappings from the contributing schemas to the integrated schema. Our method, PORSCHE (Performance ORiented SCHEma mediation), utilises a holistic approach which first clusters the nodes based on linguistic label similarity. Then, it applies a tree mining technique using node ranks calculated during depth-first traversal. This minimises the target node search space and improves performance, which makes the technique suitable for large scale data sharing.

5.1 Preliminaries

Omnipresence of XML as a data exchange format on the web and the presence of metadata available in that format forces us to focus on schema matching, and on matching for XML schemas in particular. We represent schemas as trees in our approach, as discussed in chapter 2 (Def. 2.1).

Schema matching finds similarities between elements in two or more schemas. There are three basic match cardinalities at element level [86], as discussed in chapter 4. Since we are matching schema tree structures (elements are nodes), where the leaf nodes hold data, we place more emphasis on leaf node matching. Our cate-

gorisation of node match cardinalities is driven by the node's leaf or non-leaf (inner node) status.

i) 1:1 - one node of source schema corresponds to one node in the target schema; leaf:leaf or non-leaf:non-leaf.
ii) 1:n - one node in the source schema is equivalent to a composition of n leaves in the target schema; leaf:non-leaf, where a source leaf node is mapped to a subtree containing n leaf nodes in the target.
iii) n:1 - n leaves in source schema compositely map to one leaf in the target schema; non-leaf:leaf, allowing a subtree with n leaves in a source to be mapped to a target leaf.

Example 5.1 (Match Cardinality): For 1:1, cardinality is straightforward. For 1:n, consider Figure 5.1. A match is found between $S_{source}name[2]$, child of *writer*, and $S_{target}name[2]$, child of *author*, with children *first* and *last*. We have a 1:2 mapping $(name)_{source} : (name/first, name/last)_{target}$. Also, there is a match between $S_{source}publisher[4]$ and $S_{target}publisher[5]$, with a 1:1 mapping $(publisher/name)_{source} : (publisher)_{target}$.•

Semantically speaking, a match between two nodes is fuzzy. It can be either an equivalence or a partial equivalence. In a partial match, the similarity is partial. It is highlighted in the following example.

Example 5.2 (Partial Match): In source schema Name = 'John M. Brown', is partially matched to LastName = 'Brown' and FirstName = 'John' in the target, because Name also contains the MiddleInitial = 'M'. •

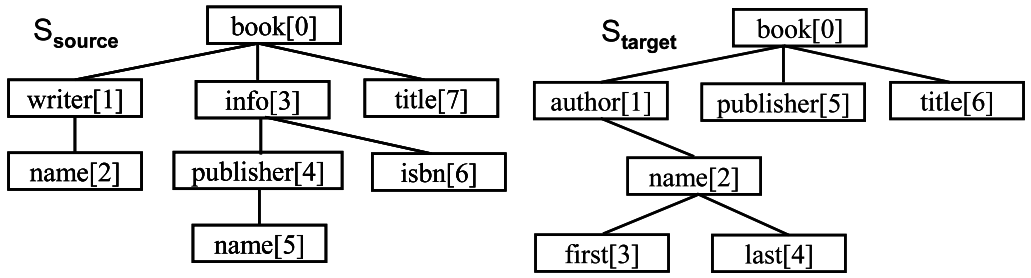

Figure 5.1: Example schema trees showing labels and depth-first order number for each node.

5.1.1 Definitions

Semantic matching requires the comparison of concepts structured as schema elements. Labels naming the schema elements are considered to be concepts and each element's contextual placement information in the schema further defines the semantics. For example, in Figure 5.1, $S_{source}name[2]$ and $S_{target}name[5]$ have similar labels but their tree contexts are different, which makes them conceptually disjoint. Considering the XML schema as a tree, with reference to schema tree definition given in chapter 2, the combination of the node label and the structural placement of the node defines the concept. To identify the node placement in the tree structure, we traverse each schema using the depth-first algorithm. We alocate an integer value to each node, as shown in Figure 5.1. Here, we present the definitions used in schema matching and integration.

Schema tree nodes bear two kinds of information: the node label and the node number allocated during depth-first traversal. Labels are linguistically compared to calculate label similarity (Def. 5.1, Label Semantics). Node number is used to calculate the node's tree context (Def. 5.3, Node Scope).

Definition 5.1 (Label Semantics): A label l is a composition of m strings, called tokens. We apply the tokenisation function *tok* which maps a label to a set of tokens $T_l = \{t_1, t_2, \ldots, t_m\}$. Tokenisation [43] helps in establishing similarity between two labels.
$tok : L \rightarrow \mathcal{P}(T)$, where $\mathcal{P}(T)$ is a power set over T.

Example 5.3 (Label Equivalence): *FirstName*, tokenised as {first, name}, and *NameFirst*, tokenised as {name, first}, are equivalent, with 100 % similarity.•

Label semantics corresponds to the meaning of the label (irrespective of the node it is related to). It is a composition of meanings attached to the tokens making up the label. As shown by Examples 5.3-5.5, different labels can represent similar concepts. We denote the concept related to label l as $C(l)$.

Example 5.4 (Synonymous Labels): *WriterName*, tokenised as {writer, name}, and *AuthorName*, tokenised as {author, name} are equivalent (they represent the same concept), since 'writer' is a synonym of 'author'.•

Semantic label matching minimises the search space of possible mappable target nodes [43, 104]. The derivation of concept similarity in two schemas is initiated

by comparing their labels. Similarity between labels is either equivalence or partial equivalence, or no similarity, as defined below.

a. Equivalence : $C(l_x) = C(l_y)$

b. Partial Equivalence : $C(l_x) \cong C(l_y)$

   i) More Specific (Is part of) : $C(l_x) \subseteq C(l_y)$

   ii) More General (Contains) : $C(l_x) \supseteq C(l_y)$

   iii) Overlaps : $C(l_x) \cap C(l_y) \neq \emptyset$

c. No Similarity : $C(l_x) \cap C(l_y) = \emptyset$

Example 5.5 (Label Similarity): As *AuthorName* and *WriterName* are equivalent (Example 5.4), we write AuthorName = WriterName. Also, AuthorLastName $\subseteq$ AuthorName, as *LastName* is conceptually part of *name*. Conversely, AuthorName $\supseteq$ AuthorLastName. *MiddleLastName* and *FirstNameMiddle* overlap, as they share tokens {name, middle}.•

Definition 5.2 (Node Clustering): To minimise the target search space (see Sec. 4.3), we cluster all nodes, based on label similarity. The clustering function can be defined as $VT : L \rightarrow \mathcal{P}(V)$ where $\mathcal{P}(V)$ is the power set over V. VT returns for each label $l \in$ L a set of nodes $v_i \subseteq V$, with labels similar to l. Figure 5.2 illustrates

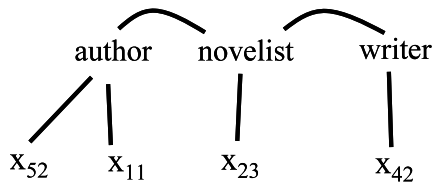

Figure 5.2: Node Clustering. In x_{sn}, s is the schema number and n is the node number within the schema.

node clustering. Here, the cluster contains nodes $\{x_{52}, x_{11}, x_{23}, x_{42}\}$ bearing synonymous labels {author, novelist, writer}.

Definition 5.3 (Node Scope): In schema S each node $x \in V$ is numbered according to its order in the depth-first traversal of S (the root is numbered 0). Let *SubTree(x)* denote the sub-tree rooted at x, and x be numbered X, and let y be the rightmost leaf (or highest numbered descendant) under x, numbered Y. Then the scope of x is *scope(x)=[X, Y]*. Intuitively, *scope(x)* is the range of nodes under x, and includes x itself, see Fig. 5.3. The count of nodes in *SubTree(x)* is $Y - X + 1$.

Definition 5.4 (Node Semantics): Node semantics of node x, C_x, combines the semantics of the node label l, $C(l_x)$, with its contextual placement in the tree, *TreeContext(x)* [43], via function *NodeSem*.
$C_x : x \rightarrow NodeSem(C(l_x),\ TreeContext(x))$.

TreeContext of a node is calculated relative to other nodes in the schema, using

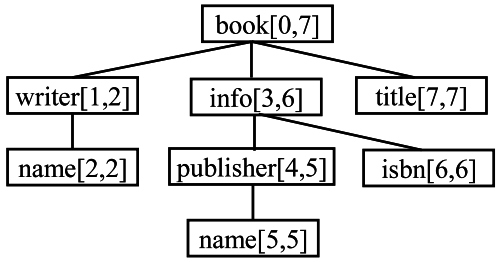

Figure 5.3: Example schema tree with labels and [number,scope] for each node.

node number and scope (Example 5.7).

Definition 5.5 (Schema Mediation)
INPUT: A set of schema trees $SSet = \{S_1, S_2, \ldots, S_u\}$.
OUTPUTS:
a) An integrated schema tree S_m which is a composition of all distinct concepts C_x in $SSet$ (see Def. 5, and[71]).

$$S_m = \bigwedge_{x \in S_i} \biguplus_{i=1}^{u} (C_x)$$

where $\biguplus$ is a composition operator on the set of schemas which produces a tree containing all the distinct concepts (encoded as nodes). The tree has to be complete (see integrated schema integrity measure, Sec. 5.3 and[71]) to ensure correct query results.
b) A set of mappings $M = \{m_1, m_2, \ldots m_w\}$ from the concepts of input schema trees to the concepts in the integrated schema. The integrated schema S_m is a composition of all nodes representing distinct concepts in *SSet*. During the integration, if an equivalent node is not present in S_m, a new edge $e\prime$ is created in S_m and the node is added to it, to guarantee completeness.

5.1.2 Scope Properties

Scope properties describe the contextual placement of a node[104]. They explain how structural context (within a tree) can be extracted during the evaluation of

node similarity. The properties represent simple integer operations.

Unary Properties of a node x with scope [X,Y]:
Property 1 (Leaf Node(x)): $X{=}Y$.
Property 2 (Non-Leaf Node(x)): $X < Y$.

Example 5.6 (Leaf and Non-Leaf) : See Fig. 5.3. Property 1 holds for *title*[7,7] which is a leaf. Property 2 holds for *writer*[1,2] which is an inner node. Intuitively, properties 1 and 2 detect simple and complex elements in a schema.•

Binary Properties for x [X,Y], $x_d[X_d, Y_d]$, $x_a[X_a, Y_a]$, and $x_r[X_r, Y_r]$:
Property 3 (Descendant (x, x_d), x_d is a descendant of x): $X_d >$X and $Y_d \leq$Y.
Property 4 (DescendantLeaf (x, x_d)): This combines Properties 1 and 3. $X_d >$X and $Y_d \leq$Y and $X_d = Y_d$.
Property 5 (Ancestor (x, x_a), x_a is an ancestor of x): (complement of Property 3) $X_a <$X and $Y_a \geq$Y.
Property 6 (RightHandSideNode (x, x_r), x_r is Right Hand Side Node of x with Non-Overlapping Scope): $X_r > Y$.

Example 5.7 (Node Relationships): See Fig. 5.4. S_{target} – property 5 holds

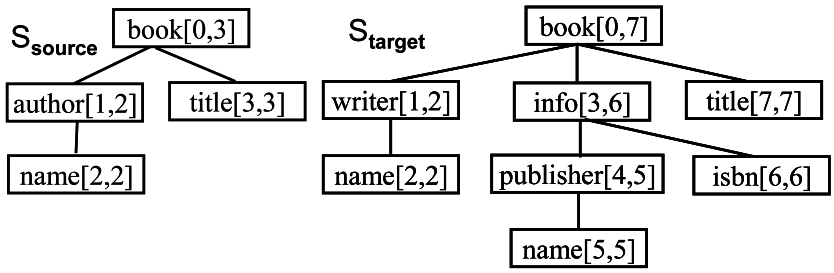

Figure 5.4: Source and target schema trees.

for nodes [4,5] and [5,5], as **Ancestor**([5,5],[4,5]), so *publisher*[4,5] is an ancestor of *name*[5,5]. Also **Ancestor**([4,5],[0,7]) holds for *book*[0,7] and *publisher*[4,5]. In S_{target}, **RightHandSideNode**([4,5],[6,6]) holds, implying node labelled *isbn* is to the right of node labelled *publisher*.•

Example 5.8 (Using Scope Properties) : The task is to find a mapping for S_{source}author/*name* in the target schema S_{target} (Fig. 5.4). S_{target} has two nodes called *name*: [2,2] and [5,5]. We assume synonymy between *author* and *writer*, top down traversal, and S_{source}*author* being already mapped to *writer* [1,2]. We per-

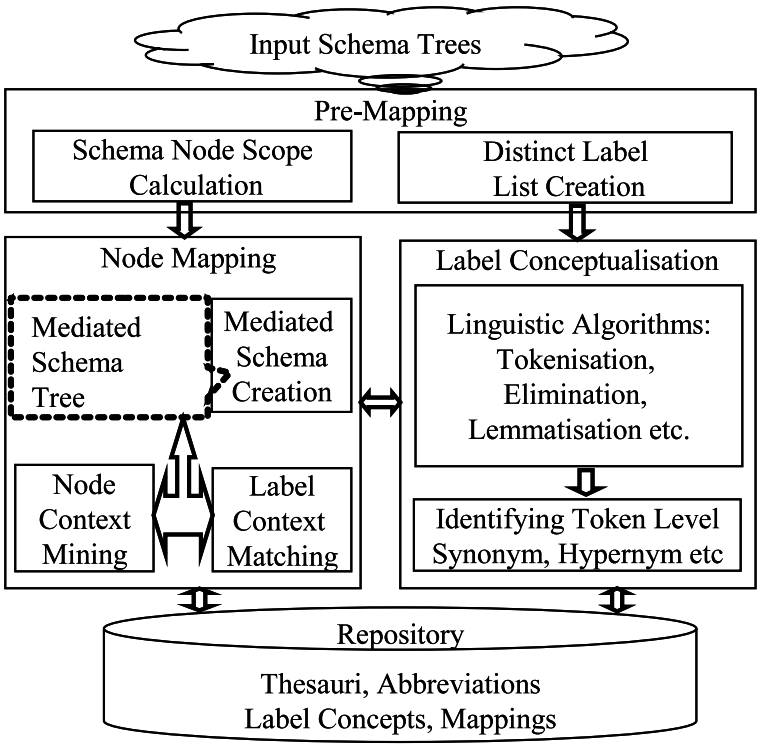

Figure 5.5: PORSCHE architecture consists of three modules: pre-mapping, label conceptualisation and node mapping.

form the descendant node check on [2,2] and [5,5] with respect to *writer*[1,2]. Using Prop. 3, **Descendant**([5,5],[1,2])=false implies [5,5] is not a descendant of [1,2], whereas **Descendant**([2,2],[1,2]) is true. Thus, [2,2] is a descendant of [1,2], and author/*name* is mapped to writer/*name*.

5.2 PORSCHE

PORSCHE is a method, which accepts a set of XML schema trees. It outputs an integrated schema tree and mappings from source schemas to the integrated schema.

5.2.1 PORSCHE Architecture

PORSCHE architecture (see Fig. 5.5) supports the complete semantic integration process involving schema trees in a large-scale scenario. The integration system is composed of three parts: i) *Pre-Mapping*, ii) *Label Conceptualisation* and iii) *Node Mapping*, supported by a repository which houses oracles and mappings.

The system is fed a set of XML Schema instances. *Pre-Mapping* module processes the input as trees, calculating the depth-first node number and scope (Def. 4) for each of the nodes in the input schema trees. At the same time, for each schema tree

a listing of nodes is constructed, sorted in depth-first traversal order. As the trees are being processed, a sorted global list of labels over the whole set of schemas is created (see Sec. 5.2.2).

In *Label Conceptualisation* module, label concepts are derived using linguistic techniques. We tokenise the labels and expand the abbreviated tokens using an abbreviation oracle. Currently, we utilise a domain specific user defined abbreviation table. Further, we make use of token similarity, supported by an abbreviation table and a manually defined domain specific synonym table. Label comparison is based on similar token sets or similar synonym token sets. The architecture is flexible enough to employ additional abbreviation or synonym oracles or arbitrary string matching algorithms.

In *Node Mapping* module, the Mediated Schema Creator constructs the initial mediated schema from the input schema tree with the highest number of nodes augmented with a virtual root. Then it matches, merges and maps. Concepts from input schemas are matched to the mediated schema. The method traverses each input schema depth-first, mapping parents before siblings. If a node is found with no match in the mediated schema, a new concept node is created and added to the mediated schema. It is added as the rightmost leaf of the node in the mediated schema to which the parent of the current node is mapped. This new node is used as the target node in the mapping. The technique combines node label similarity and contextual positioning in the schema tree, calculated with the help of properties defined in Section 5.1.

The *Repository* is an indispensable part of the system. It houses oracles: thesauri and abbreviation lists. It also stores schemas and mappings, and provides persistent support to the mapping process.

5.2.2 Algorithms and Data Structures

In this section, we discuss the hybrid algorithms used in PORSCHE. We make assumptions presented in [50, 96] which hold in single domain schema integration. Schemas in the same domain contain the same domain concepts, but differ in structure and concept naming, for instance, *name* in one schema may correspond to a combination of *FirstName* and *LastName* in another schema. Also, in one schema different labels for the same concept are rarely present. Further, only one type of match between two labels in different schemas is possible, for example, *author* is a synonym of *writer*.

Pre-Mapping comprises a number of functions: 1) *depth-first, scope and parent node number calculation*, 2) creation of *data structures for matching and integration*:

Algorithm : preMap

Data: *SSet : Set of Schema Trees of size u*
Result: $\mathscr{V}$, $\mathscr{GLL}$, j
$\mathscr{V}$: List of lists of nodes (one list per schema) of size u, initially empty; each node list $\mathscr{SNL}$ sorted on depth-first order
$\mathscr{GLL}$: Global sorted list of labels
j : Schema tree identifier, initialized to 0

```
1  begin
2      for each schema S_i ∈ S do
3          V_i ← nodeScope(∅, RootNode_{S_i}, ∅)
4          𝒱 ← 𝒱 ∪ V_i
5          L_i ← lab(V_i)
6          if i = 1 then
7              𝒢LL ← sort(L_i)
8          else
9              𝒢LL ← mergeLabelLists(L_i,𝒢LL)
10     j ← initialMediatedSchema(𝒱_random|𝒱_smallest|𝒱_largest)
11 end
```

Figure 5.6: Pseudocode of Pre-Mapping.

schema node list ($\mathscr{SNL}$) for each schema and global label list ($\mathscr{GLL}$), and 3) *identification of the input schema* from which the initial mediated schema is created. $\mathscr{SNL}$ and $\mathscr{GLL}$ are later updated by the Node Mapping module (see Fig. 5.11). Pre-Mapping requires only one traversal of each input schema tree (Fig. 5.6).

Algorithm : nodeScope

Data: p, c, V
p : *parent node list element,* c : *current node,* V : *nodes list*
Result: V

```
1  begin
2      x ← New nodesListElement(c)
3      x.number ← length(V)
4      x.parentNode ← p
5      x.rightMostNode ← ∅
6      Add x to V
7      if c has no children then
8          update(x, x)
9      for each child of c do
10         nodeScope(x, child, V)
11 end
```

Figure 5.7: Pseudocode for node scope calculation and node list creation.

Algorithm *nodeScope* (Fig. 5.7) is a recursive method, called from the *preMap* algorithm (L6.3)[1] for each schema, generating a schema node list ($\mathscr{SNL}$[2]). It takes as input the current node, its parent reference, and the node list. In its first activation,

[1] Lx.y refers to line y of algorithm in figure x
[2] SNL: Schema Nodes List

```
Algorithm : update
  Data: x_c, x_r
  x_c : Current nodes list element
  x_r : Right most node element of current nodes list element
1 begin
2 |  x_c.rightMostNode ← x_r
3 |  if x_c ¬ Root Node then
4 |  |  update(x_c.parentNode, x_r)
5 end
```

Figure 5.8: Pseudocode for updating scope entries in $\mathscr{S}NL$.

reference to the schema root is used as the current node: there is no parent, and $\mathscr{S}NL$ is empty. A new node list element is created (L7.2-6) and added to $\mathscr{S}NL$. Next, if the current node is a leaf, a recursive method *update* (Fig. 5.8) is called. This adjusts the rightmost node reference for the current $\mathscr{S}NL$ element and then goes to adjust its ancestor entries in $\mathscr{S}NL$. This method recurses up the tree till it reaches the root, and adjusts all elements on the path from the current node to the root. Next, in *nodeScope* (L7.9-10), the method is called for each child of the current node.

After calculating node scope, *preMap* adds the new $\mathscr{S}NL$ to its global list of schemas and creates a global (sorted) label list ($\mathscr{G}LL$) (L6.5-9). (L6.10) chooses the largest (smallest or random) schema tree for subsequent formation of the initial mediated schema. $\mathscr{G}LL$[3] creation is a binary incremental process. $\mathscr{S}NL$ of the first schema is sorted and used as an initial $\mathscr{G}LL$ (L6.6-7) and then $\mathscr{S}NL$s from other schemas are iteratively merged with the $\mathscr{G}LL$, as follows.

mergeLabelLists (see Fig. 5.9) is a variant of merge sort. At (L9.7-8), we skip labels shared between $\mathscr{S}NL$ and $\mathscr{G}LL$ and add links between them, to keep track of shared labels. Multiple occurrences of the same label within an $\mathscr{S}NL$, however, are needed, as they help us in identifying labels attached to distinct nodes, e.g. author/*name* is different from publisher/*name*. If more distinct nodes with the same label are encountered in the matching process, they are handled in an overflow area. In the $\mathscr{G}LL$, multiple occurrences of the same label are equivalent to the largest number of their separate occurrences in one of the input schemas. This is further explained with the help of the following example.

Example 5.9 (Repeated Labels) : Assume three input schema trees: S_1, S_2 and S_3. Their $\mathscr{S}NL$s before merge sort are: S_1\{book, author, name, info, publisher,

[3]GLL: Global Label List

```
Algorithm : mergeLabelLists
   Data: L1, L2 : Label lists for merging
   Result: GLL : Sorted list of merged label lists, initially empty
 1 begin
 2     sort(L1)
 3     while length(L1) ≥ 0 ∧ length(L2) > 0 do
 4         if first(L1) ≤ first(L2) then
 5             append first(L1) to GLL
 6             L1 ← rest(L1)
 7             if first(L1) = first(L2) then
 8                 L2 ← rest(L2)
 9         else
10             append first(L2) to GLL
11             L2 ← rest(L2)
12     if length(L1) > 0 then
13         append rest(L2) to GLL
14     if length(L2) > 0 then
15         append rest(L1) to GLL
16 end
```

Figure 5.9: Pseudocode for label list merging, based on merge sort.

```
Algorithm : labelSimilarity
   Data: GLL, simType
   GLL : Global label list
   simType : Similarity type for labels
   Result: GLL : Global label list with inter-label similarity links
 1 begin
 2     var Set of token sets T
 3     (T,GLL) ← tok(GLL)
 4     if simType = synonymTokenSetSimilarity then
 5         adjustTokenSynonymSimilarity(Union of token sets ∈ T)
 6     for each li ∈ GLL do
 7         for each lj ∈ rest(GLL) do
 8             if li ¬= lj ∧ ∄ similarity(li,lj) then
 9                 if Ti = Tj then
10                     adjustLabelSimilarity(li,lj)
11 end
```

Figure 5.10: Pseudocode for label similarity derivation, based upon tokenisation (tok).

name, title}, S_2{book, writer, name, publisher, address, name, title} and S_3{book, author, address publisher, address, name, title}. The sorted $\mathscr{G}LL$ is {address, address, author, book, info, name, name publisher, title, writer}.•

Label conceptualisation is implemented in *labelSimilarity* (Fig. 5.10). The method

```
Algorithm : mediation
  Data: 𝒱, j
  𝒱 : List of lists of nodes (one list per schema) of size u; each node list
  𝒮NL sorted on depth-first order;
  j : Input schema identifier
  Result: M, V_m
  M : Set of mapping, initially empty
  V_m : Mediated schema nodes list
1 begin
2   V_m ← initialMediatedSchemaNodesList(V_j, x_ROOT)
3   for each V ∈ 𝒱 do
4     for each node x ∈ V do
5       L_x ← lab(x)
6       L_xsl ← similarLabels(L_x, V_m)
7       if L_xsl ¬ ∅ then
8         V_t ← VT(L_xsl)
9         if |V_t| = 1 then
10          (m, xt) ← oneMatch(x, V_t, V_m)
11        else
12          (m, xt) ← multipleMatch(x, V_t, V_m)
13      else
14        (xt, V_m) ← addNewNode(V_m, x)
15        (m, xt) ← mapOneToOne(x, xt)
16      M ← M ∪ m
17 end
```

Figure 5.11: Pseudocode for schema integration and mapping generation.

creates similarity relations between labels, based on label semantics (Def. 2). This method traverses all labels, tokenises them (with abbreviations expanded), and compares them exhaustively. If similarity is detected, a link is added in $\mathscr{G}LL$ recording label similarity between two list entries.

After Pre-Mapping phase, PORSCHE carries out Node Mapping (see Fig. 5.11, mediation), which accomplishes both schema matching and integration. This handles the semantics of node labels, along with the tree context, to compute the mapping. The algorithm accepts as input a list of $\mathscr{S}NL$s ($\mathscr{V}$) and the identifier of the input schema with which other schemas will be merged (j). It outputs the mediated schema node list ($\mathscr{M}SNL$[4]), V_m, and a set of mappings M, from input schema nodes to the mediated schema nodes.

First, the initial mediated schema tree which works as a seed for the whole process is identified. To this purpose, a clone of the largest $\mathscr{S}NL$ is created with a new ROOT node, V_m (L11.2). The new root node allows us to add input schema trees at the top level of the mediated schema whenever our algorithm does not find any similar nodes. This is required to support the completeness of the final mediated

[4]MSNL: Mediated Schema Nodes List

schema.

We use three data structures mentioned in the preceding paragraphs: (a) input $\mathscr{S}NL$s, $\mathscr{V}$ (b) $\mathscr{G}LL$, and (c) $\mathscr{M}SNL$, V_m. Here, we give a brief explanation of attributes associated with the node objects in each data structure. $\mathscr{V}$ holds u $\mathscr{S}NL$s, representing each input schema, where each element of the list has attributes *{depth-first order number, scope, parent, mapping data {mediated schema node reference, map type}}*. $\mathscr{G}LL$ element attributes are *{label, link to similar label}*. V_m's elements comprise attributes *{depth-first order number, scope, parent, list of mappings{input schema identifier, node number}}*. Data structures (a) and (c) are further sorted according to the order of (b), i.e. sorted $\mathscr{G}LL$, where each node object is placed aligned with its label (see Tab. 5.1). This helps in the clustering of similar nodes (Def. 3) and speeds up mapping, as similar nodes can be looked up via an index lookup. In *mapping data* in $\mathscr{V}$, the *mediated schema node reference* is the index number of the node in $\mathscr{M}SNL$ to which it has been matched, and *maptype* records mapping cardinality (1:1, 1:n or n:1).

Algorithm : oneMatch

```
   Data: x, V_t, V_m
   x : Source node
   V_t : Set of one target node xt_1
   V_m : Mediated schema tree node list
   Result: m, xt
   m : Mapping
   xt : Target node in V_m
 1 begin
 2   if (Leaf(x)∧Leaf(xt_1)) ∨ (¬Leaf(x)∧¬Leaf(xt_1)) then
 3     if ancestorMap(x,xt_1) then
 4       (m, xt)← mapOneToOne(x, xt_1)
 5     else
 6       (xt, V_m) ← addNewNode(V_m,x)
 7       (m, xt)← mapOneToOne(x, xt)
 8   if Leaf(x)∧¬Leaf(xt_1) then
 9     (m, xt)← mapOneN(x, xt_1)
10   if ¬Leaf(x)∧Leaf(xt_1) then
11     (m, xt)← mapNOne(x, xt_1)
12 end
```

Figure 5.12: Pseudocode for matching one target node.

During mediation a match for every node (L5.11.4) of each input schema (L5.11.3) is calculated, mapping it to the mediated schema nodes. For each input node x, a set V_t of possible mappable target nodes in the mediated schema is created, producing the target search space for x. The criterion for the creation of this set of nodes is node label equivalence or partial equivalence (L5.11.5,6). V_t can have zero

```
Algorithm : multipleMatch
  Data: x, V_t, V_m
  x : Source node
  V_t : Set of n(> 1) target nodes
  V_m : Mediated schema tree node list
  Result: m, xt
  m : Mapping
  xt : Target node in V_m
1 begin
2   descendantCheck ← false
3   for each node xt_i ∈ V_t do
4     if descendant(x,xt_i) then
5       (m, xt) ← mapOneToOne(x, xt_i)
6       descendantCheck ← true
7       break
8   if ¬descendantCheck then
9     (xt, V_m)← addNewNode(V_m,x)
10    (m, xt) ← mapOneToOne(x, xt)
11 end
```

Figure 5.13: Pseudocode for matching several target nodes.

(L5.11.13), one (L5.11.9) or several (L5.11.11) nodes.

If there is only one possible target node in the mediated schema, method *one-Match* (Fig. 5.12) is executed. (L5.12.2,8,10) compare the tree context of nodes x (input node) and xt_1 (possible target node in V_t), to ensure that a leaf node is mapped to another leaf node. Second check, *ancestorMap* (L5.12.3), ensures that at some point up the ancestor hierarchy of x and xt_1 there has been a mapping, in accordance with Prop. 5. This guarantees that subtrees containing x and xt_1 correspond to similar concept hierarchies. This increases the match confidence of nodes x and xt_1.

Alternatively, if the target search space V_t has more than one node, algorithm *multipleMatch* (Fig. 5.13) is executed. Here, we check the descendant Prop. 3 (L5.13.4) for each node xt in V_t (L5.13.3). *Descendant* function verifies if xt lies in the sub-tree of the node to which the parent is mapped, that is within the scope. The function returns true for only one node or none.

Method *addNewNode* (L5.11.14,5.13.9,5.12.6) adds a new node xt as the right-most sibling of the node to which the parent of x is mapped. Adding a new node requires a scope update: the ancestor and right hand side nodes of the new node have now a larger scope. Properties 5 and 6 are used to to find the ancestor and right hand side nodes. For ancestor nodes, scope is incremented, and for right hand side nodes, both node number and scope are incremented.

Mapping method *mapOneToOne*, creates a 1:1 map from x to xt, whereas *mapO-*

neN creates a mapping from x (a leaf) to a set of leaves in the subtree rooted at xt (non-leaf), which is a 1:n mapping. This mapping is considered to be an approximate mapping but follows the semantics of leaves mapping to leaves. And, similarly, *mapNOne* is a composite mapping of leaves under x, to leaf xt. Node Mapping algorithm (Fig. 5.11) integrates all input nodes into the mediated schema and creates corresponding mappings from input schemas to the mediated schema.

5.2.3 A Schema Integration Example

Figure 5.14 shows two trees after Pre-Mapping. A list of labels created in this traversal is shown in Table 5.1a. The two nodes with the same label *name* but different parents are shown (labels with index 2 and 3). The last entry in the list is the ROOT of the mediated schema. In the label list the semantically similar (equivalent or partially equivalent) labels are detected: *author* (index 0) is equivalent to *writer* (index 7).

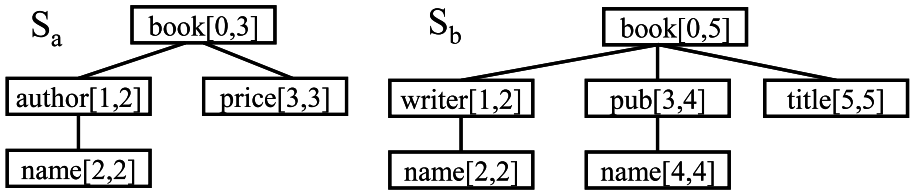

Figure 5.14: Input Schema Trees S_a and S_b.

Table 5.1: Before Node Mapping. Schema column entry is (node number, scope, parent).

a. Global Label List

	0	1	2	3	4	5	6	7	8
	author	book	name	name	price	pub	title	writer	ROOT

b. Input Schema Matrix

Sa	1,2,0	0,3,-1	2,2,1		3,3,0				
Sb		0,5,-1	2,2,1	4,4,3		3,4,0	5,5,0	1,2,0	

c. Initial Mediated Schema

		1,6,0	3,3,2	5,5,4		4,5,1	6,6,1	2,3,1	0,6,-1

A matrix of size uq (here 2x9) is created, where u is the number of schemas and q the number of distinct labels in the $\mathscr{G}LL$, see Tab. 5.1b. Each matrix row represents an input schema tree and each non-null entry contains the node scope, parent node link, and the mapping, which is initially null. Matrix columns are alphabetically ordered. The larger schema, S_b, Fig. 5.14, is selected for the creation of the initial

Table 5.2: After Node Mapping. *Column entry is (node number, scope, parent, **mapping**). **Bold numbers refer to **sourceSchema.node**.

a. Global Label List

	0	1	2	3	4	5	6	7	8
	author	book	name	name	price	pub	title	*writer*	ROOT

b. Mapping Matrix *

Sa	1,2,0,**7**	0,3,-1,**1**	2,2,1,**2**		3,3,0,**4**				
Sb		0,5,-1,**1**	2,2,1,**2**	4,4,3,**3**		3,4,0,**5**	5,5,0,**6**	1,2,0,**7**	

c. Final Mediated Schema **

		1,7,0, **1.0,2.0**	3,3,2, **1.2,2.2**	5,5,4, **2.4**	7,7,1, **1.3**	4,5,1, **2.3**	6,6,1, **2.5**	2,3,1, **1.1,2.1**	0,7,-1

mediated schema S_m. A list of size q, Tab. 5.1c, is created to hold S_m, assuming the same label order as in Tab. 5.1a and 5.1b.

The node mapping algorithm takes the data structures in Tab. 5.1 as input, and produces mappings shown in Tab. 5.2b and the integrated schema in Tab. 5.2c. In the process, the input schema S_a is mapped to the mediated schema S_m, i.e. extended schema S_b. The mapping is taken as the column index (Tab. 5.2b, in bold) of the node in S_m. Saving mappings as column index gives us the flexibility to add new nodes to S_m, by appending to it. Scope values of some nodes may be affected, as explained previously, because of the addition of new nodes, but column indices of all previous nodes remain the same. Intuitively, none of the existing mappings are affected.

Node Mapping for input schema S_a (Tab. 5.1b row 1) starts from the root node S_a[0,3]*book*. It follows the depth first traversal of the input tree, to ensure that parent nodes are mapped before the children. S_a[0,3] has only one similar node in the mediated schema tree S_m i.e., node 1 at index 1. So entry **1** at index 1 for S_a records the mapping ($S_a[0,3], S_m[1,6]$), see Tab. 5.2b. Information regarding mapping is also saved in the mediated schema node as **1.0** (node 0 of schema 1). Next node to map in S_a is [1,2]*author*, similar to S_m[1,2]*writer*. Both nodes are internal and the method *ancestorMap* returns true since parent nodes of both are already mapped. The resulting mapping for node with label *author* is entry **7**. For node with label 2 *name*, there are two possibilities, label 2 (index 2) and label 3 (index 3) in the mediated schema. *Descendant* is true for node at index 2 (*author/name,writer/name*), and false for 3 (*author/name,pub/name*). Hence, **2** is the correct match.

The last node to map in S_a is [3,3]*price*. There is no node in S_m with a similar label, so a new node is added to S_m, recorded by an entry in the column with label 'price' in the mediated schema (Tab. 5.2c). A new node is created as the

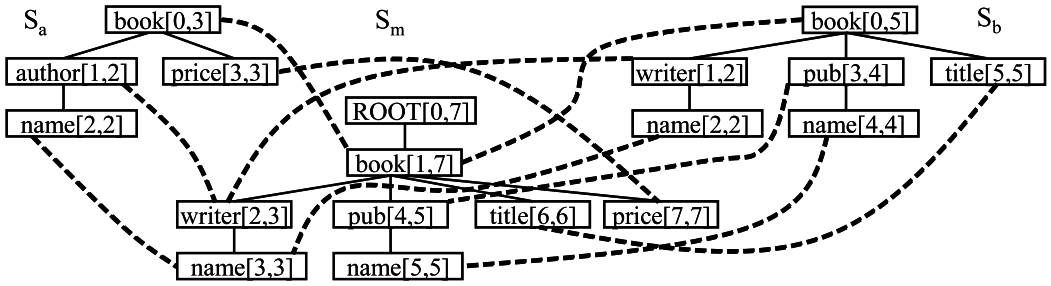

Figure 5.15: Mediated Schema with mappings.

rightmost sibling of the node in the mediated tree to which the parent node of current input node is mapped, i.e. as child of *book*. The scope and parent are accordingly adjusted for the new node, and for its ancestors, and for right hand side nodes, scope and number are adjusted. There is no effect on the existing mapping information. Finally, a mapping is created from the input node to this new target node.

If a new node is added in the middle of the tree, its ancestor's scope is incremented by one. And, accordingly, right hand side nodes (satisfying Property 6) have their order numbers and scopes incremented by one. The implementation keeps track of the columns for the next node according to depth-first order. Thus, the final mediated schema tree can be generated from the final mediated schema row by a traversal starting from the ROOT. The mediated schema with mappings (dotted lines) is shown in Fig. 5.15.

5.2.4 Complexity Analysis

The worst-case time complexity of PORSCHE can be expressed as a function of the number of nodes in an input schema, n on average, where the number of schemas is u. The algorithm has the following phases.

- Data structure creation - all input schemas are traversed as trees (depth-first) and stored as node lists ($\mathscr{S}NLs$), a global list of node labels ($\mathscr{G}LL$) is created, and one of the schemas is selected as the base for the mediated schema, with time complexity $O(nu)$.

- Label List sorting - the number of labels is nu and sort time complexity for the global list is $O(nu \log(nu))$.

- Label similarity - we compare each label to the labels following it in the list. Time complexity of $O((nu)^2)$.

- Node contextual mapping - nodes are clustered, based on node label similarity. Worst case is when all labels are similar and form one cluster of size nu. We compare each node once to all other nodes, using tree mining functions. At worst, nu nodes are compared to nu nodes, with time complexity $O((nu)^2)$. Realistically, there are multiple clusters, much smaller than nu, which improves performance.

Overall time complexity is $O((nu)^2)$, with space complexity $O(n(u)^2)$, as the largest data structure is a matrix of size nu^2 used in the schema mapping process.

The theoretical time and space complexity of PORSCHE is similar to other algorithms we surveyed (Sec. 2). Because we perform only one full traversal of all the schemas and reuse the memory-resident data structures generated in the traversal of the fist schema, instead of performing repeated traversals used in alternative approaches which use a script to repeatedly merge any two schemas, we can generate a mediated schema faster than other approaches. Our experiments confirm the complexity figures derived above, in next section.

5.3 Experimental Evaluation

We examine both the time performance and quality of schema mediation, with respect to mapping quality and mediated schema integrity.

Table 5.3: Schema domains used in the experiment.

Domain	OAGIS	XCBL	BOOKS
Number of Schemas	80	44	176
Average nodes per schema	1047	1678	8
Largest schema size	3519	4578	14
Smallest schema size	26	4	5

5.3.1 Performance Evaluation

Performance is evaluated as the number of schemas or nodes processed versus the time required for matching, merging and mapping. We selected three sets of schema trees from different domains, shown in Tab. 5.3. OAGIS and xCBL are real schemas, whereas BOOKS are synthetic schemas.

Experiments were performed with different numbers of schemas (2 to 176). Algorithm performance was timed under three different similarity scenarios:

A) Label String Equivalence,

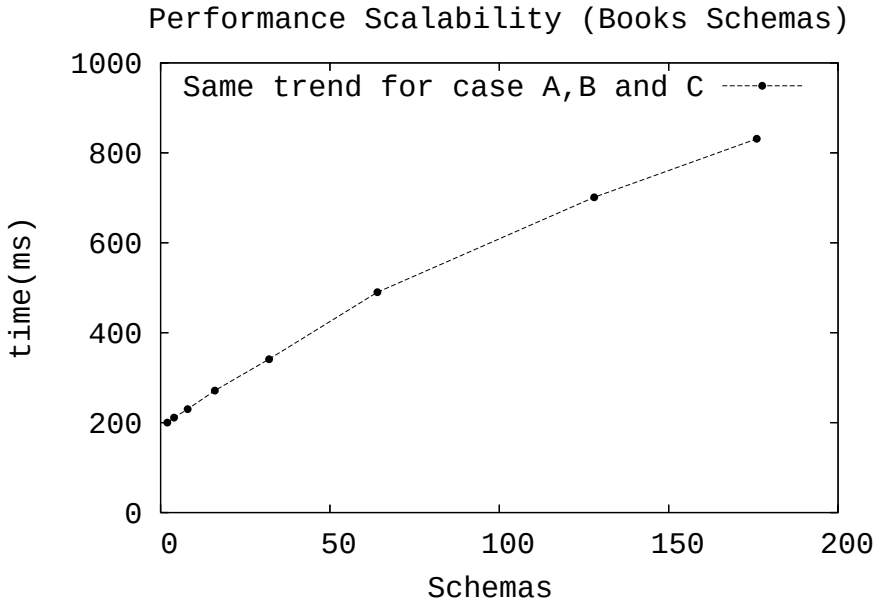

Figure 5.16: BOOKS: integration time (ms) as a function of the number of schemas.

B) Label Token Set Equivalence,

C) Label Synonym Token Set Equivalence.

Figure 5.16 shows a comparison of three similarity scenarios, A, B, and C, for sets of 2, 4, 8, 16, 32, 64, 128, and 176 schemas from BOOKS. There is no visible difference in the performance of various matchers. This is possibly due to the fact that synthetic schemas vary little in their labels.

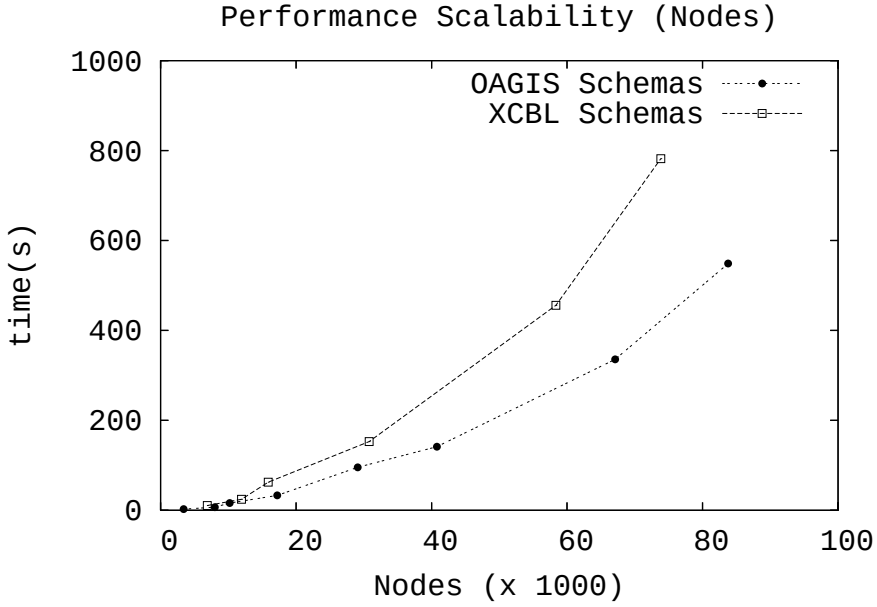

Figure 5.17: Comparison of schema integration times (seconds) for real web schemas.

Figure 5.17 shows time in seconds for OAGIS and xCBL. The execution time for PORSCHE depends upon the number of schemas to be integrated, and appears to be quadratic in the number of nodes, as predicted by the complexity analysis (Sec. 4.4).

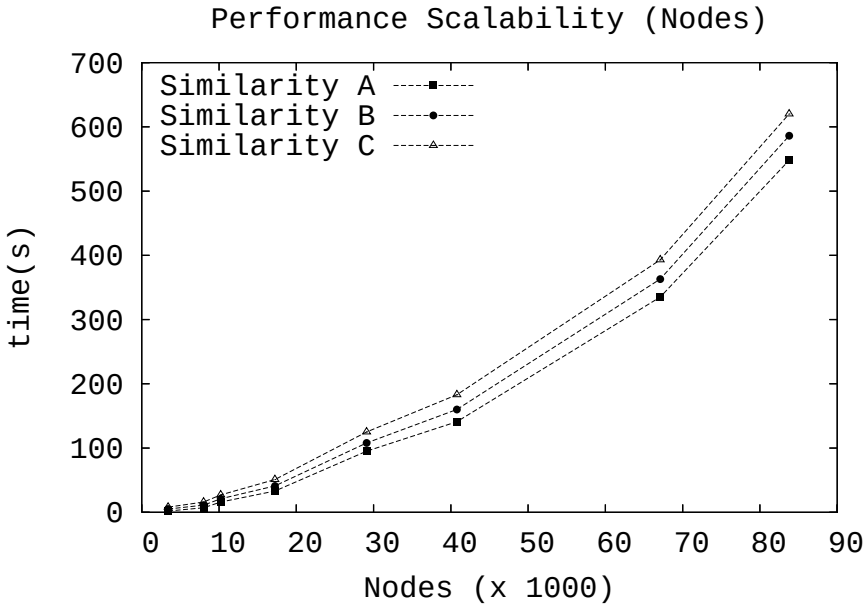

Figure 5.18: Integration of OAGIS schemas.

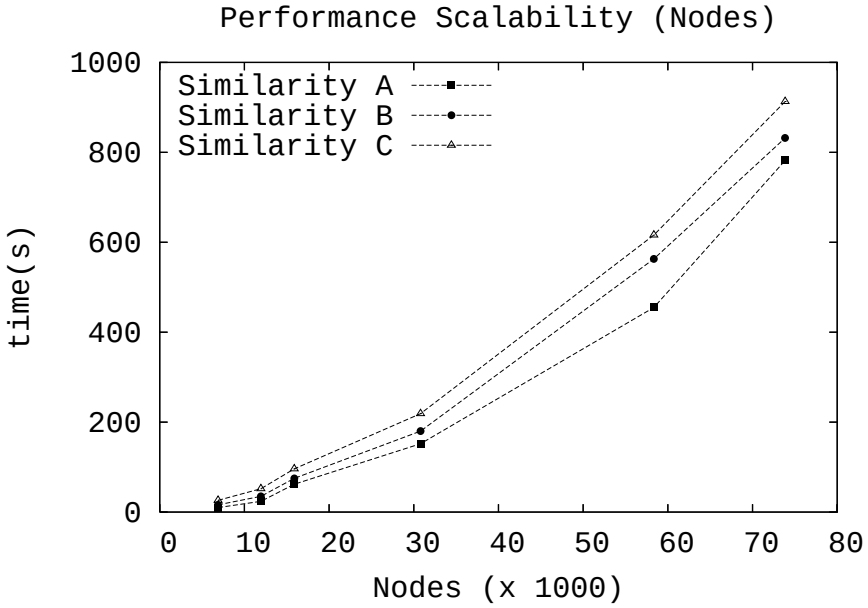

Figure 5.19: Integration of xCBL schemas.

Figures 5.18 and 5.19 show the time in seconds against the number of nodes processed for the three similarity methods (A, B, and C), for xCBL and OAGIS schemas. xCBL schemas (Fig. 5.19) are slower to match than OAGIS schemas (Fig. 5.18). This is

due to the higher average number of nodes in xCLB schemas. It takes approximately 600 s to match 80 OAGIS schemas, while 44 xCLB schemas require about 800 s.

In both cases there is a slight difference in matching times for categories A, B and C (Fig. 5.18 and 5.19), due to different label matching strategies. A is the fastest, as it works only on the labels. B is slightly slower, as labels have to be tokenised, and C is the slowest, as one needs to match synonyms as well. These evaluation cases show that PORSCHE has acceptable performance on an office PC for a large number of schemas.

5.3.2 Mapping Quality

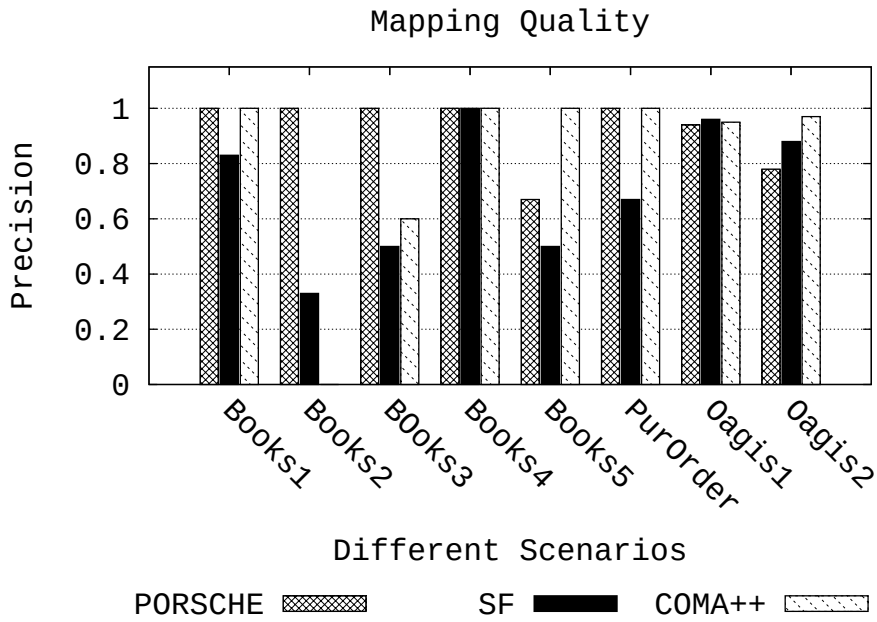

Figure 5.20: Precision for PORSCHE, COMA++ and Similarity Flooding for 8 pairs of schemas.

As available benchmarks focus on two schemas at a time, and our approach targets a set of schemas, we can only compare mapping quality for two schemas at a time. From the set of 176 books schemas we randomly selected 5 pairs of schemas (sizes ranging between 5 and 14 nodes), from purchase order one schema pair (schema sizes 14 and 18 nodes), and from OAGIS two schema pairs (schema sizes from 26 to 52 nodes)[5]. We computed mapping quality for COMA++ and SF:Similarity Flooding (RONDO) and compared those to PORSCHE. The quality of mappings is evaluated using precision, recall and F-measure. The results are summarised in Figures 5.20, 5.21, 5.22 and 5.23. For PORSCHE, similarity uses token level synonym lookup. The comparison shows almost equivalent mapping quality for

[5]Schemas and results detail at http://www.lirmm.fr/PORSCHE/results/

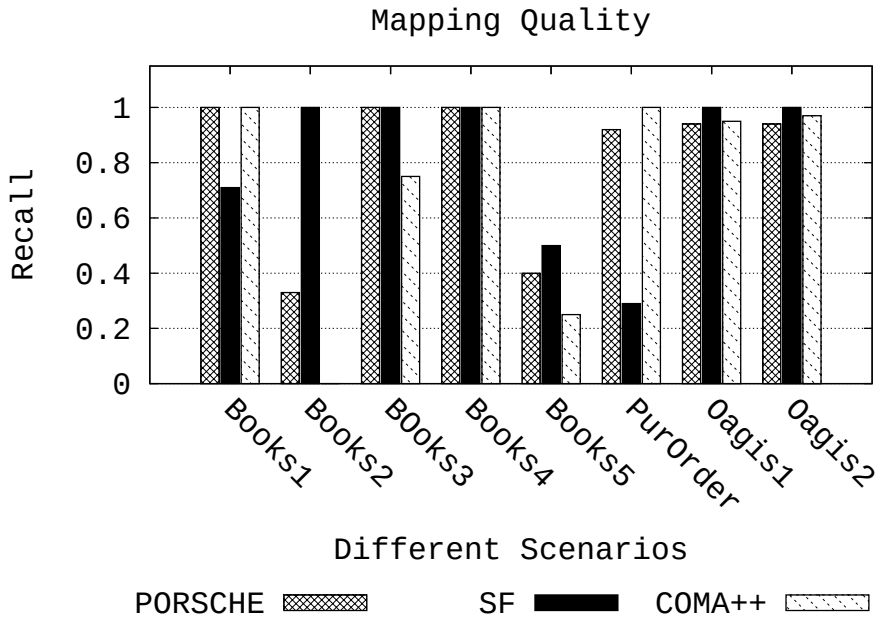

Figure 5.21: Recall for PORSCHE, COMA++ and Similarity Flooding for 8 pairs of schemas.

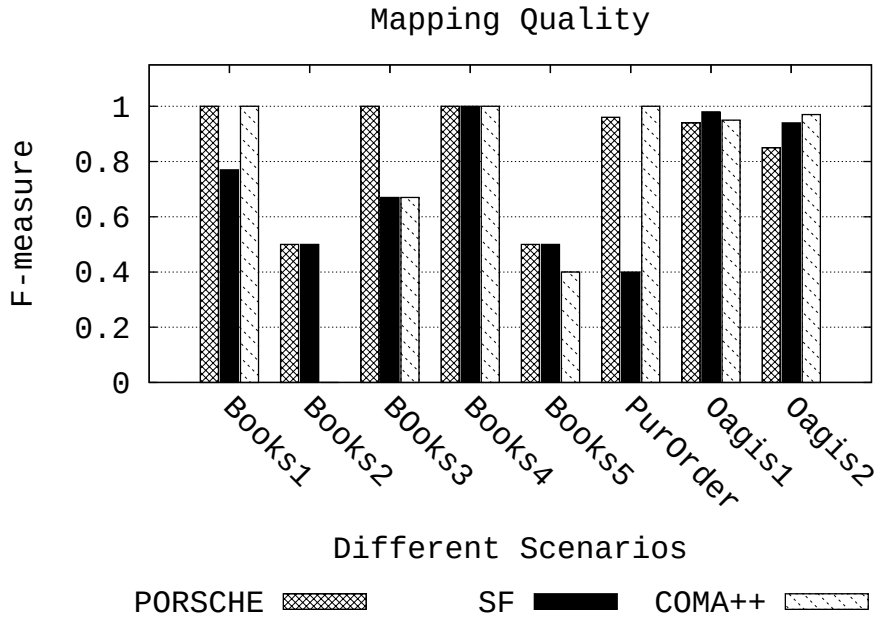

Figure 5.22: F-measure for PORSCHE, COMA++ and Similarity Flooding for 8 pairs of schemas.

PORSCHE and COMA++, since both are driven by similar user defined pre-match effort of constructing synonym tables used to derive label similarity. Similarity flooding demonstrates better recall than PORSCHE or COMA++. The results also demonstrate local schema differences. Books2 is the hardest case, and we discovered that it contained inverted paths (book/author is matched to author/book).

PORSCHE has also been evaluated within our benchmark, XBenchMatch[29].

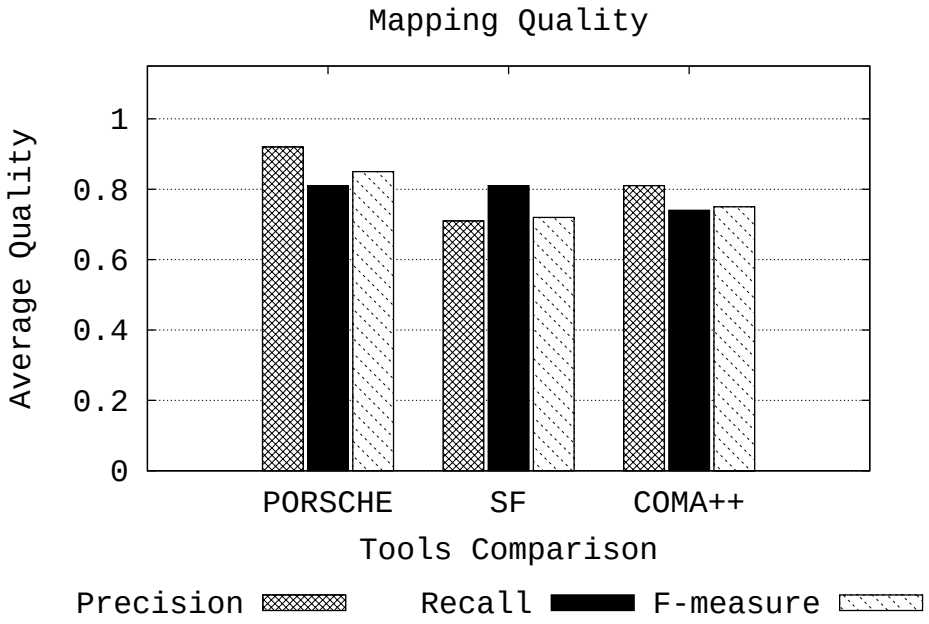

Figure 5.23: Global comparison of quality metrics.

The benchmark uses precision, recall, F-measure, overall measure, and other metrics of structural schema proximity. Structural proximity measures are based on differences in the number and size of sub-trees shared between input and mediated schemas, and schema proximity compares two schema trees which are being matched. PORSCHE demonstrated good results in comparison with COMA++ and RONDO (Similarity Flooding).

5.3.3 Mediated Schema Integrity

We gave a brief overview of the integrated schema quality evaluation in Sec. 4.3. Since our test domains are XML schema instances with only element label information, type consistency can not be evaluated. In the following, we consider only completeness and minimality. As discussed in Sec. 2.2 and algorithm implementation (Fig. 5.11, 5.12 and 5.13), when a match is not found in the mediated schema, a new concept node is added to it. Mapping from source schema tree to the new node is then established. This demonstrates that PORSCHE inherently fulfills the completeness criterion of mediated schema integrity. To evaluate minimality, we take the pairs of schemas considered in mapping quality evaluation. For each pair of schemas we perform matching and merging. The resulting integrated schema is scrutinised manually for redundancies. The results ($minimality = 1 - (redundantNodes/totalNodes)$) are shown in Fig. 5.24. An inspection of integrated schemas showed that minimality decreased where input schemas being matched had inverted paths. To further investigate the integrity

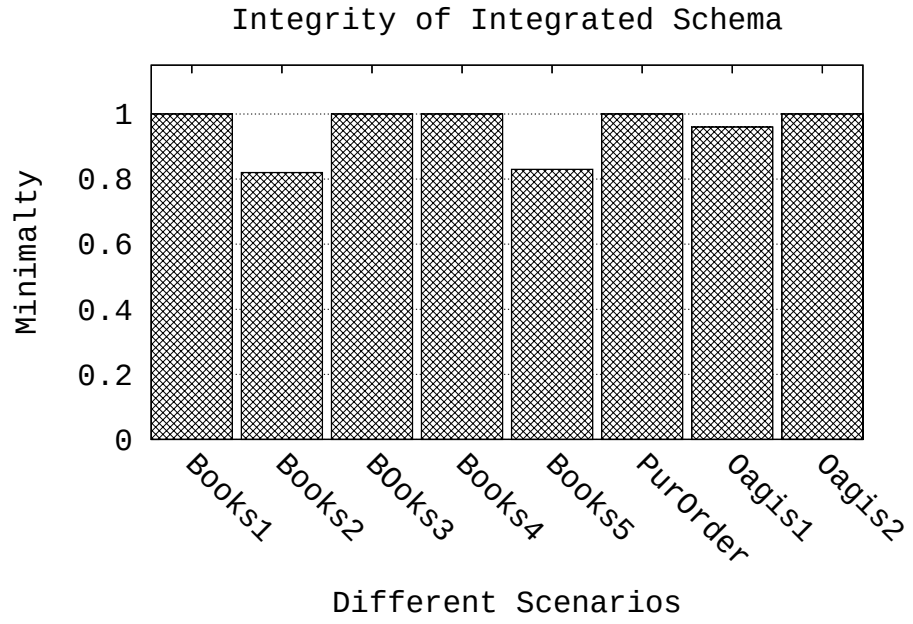

Figure 5.24: Minimality of PORSCHE schema integration for the selected pairs of schemas.

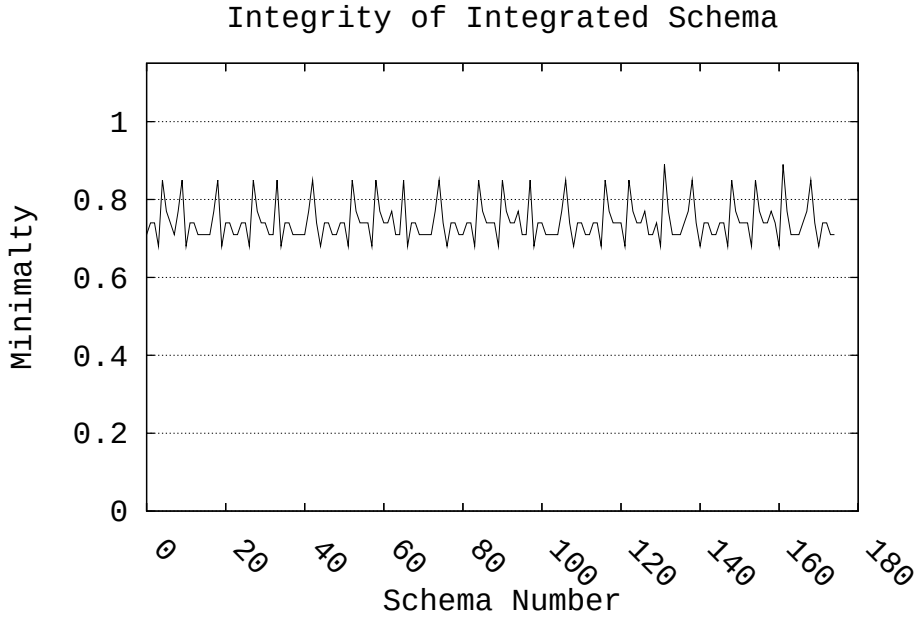

Figure 5.25: Minimality in PORSCHE for 176 book schemas. 176 experiments are shown (along the x-axis), with each schema selected, in turn, as the initial mediated schema.

of our approach, we calculated the minimalty of the integrated schema created by merging 176 books schemas (small in size and amenable to human inspection). We used an ideal integrated schema (manually created, containing 17 nodes) to assess redundancy. A batch of 176 experiments was performed, each selecting one schema as the seed mediated schema. Figure 5.25 shows that minimality fluctuates between 0.68 and 0.85.

5.4 Comparison to other Tools

Most schema-matching systems compare two schemas at a time and aim for quality matching but require significant human intervention. CUPID[66], COMA++[24], S-Match[43], QOM[32], GLUE[27] are some of these tools presented in this section. Several surveys[24, 86, 93] argue that extending the matching to data integration is time consuming and limited in scope. Matching of two large bio-medical taxonomies has been demonstrated by Do and Rahm using COMA++, and Mork and Bernstein [78] using CUPID and similarity flooding (RONDO[75]). Quick Ontology Matching (QOM) matches large ontologies with performance in view. Large scale schema matching has also been investigated in the web interface schema integration[50, 96] using data mining. PORSCHE's goal is to match, merge and map in a hybrid manner, whereas most of the tools separate the two activities of matching and integrating, which makes them unsuitable for automated integration scenarios, as needed in e-commerce.

COMA++[24] is one of the most recent matching and merging tools. It is a composite matcher producing quality matches for a pair of schemas. It provides an extensible platform which can combine several matchers. It uses user defined synonym and abbreviation tables as a pre-mapping effort. It has a comprehensive interface for navigating through the matches produced by the software. The user verifies and selects or edits the match results to finalise the mapping between the two schemas. Based on these mappings, COMA++ can also generate a merged schema, which has to be validated by the user. The authors of COMA++ give a comprehensive evaluation of match quality results but do not quantify the quality of the merged schema. In large scale schema matching, COMA++ requires a significant amount of human intervention. First, the user should identify fragments in the two schemas to be mapped. This aims to manage the namespace/ include characteristics of XML schemas. Then, individual fragments from the source schemas are mapped to fragments of the target schema, one by one, and match results are saved in a relational database for subsequent comparison. COMA++ uses a bottom up hierarchical approach: if leaves are similar, then parent nodes may also be similar. In PORSCHE we use a reverse approach: given similarity between parent or ancestor nodes, we discover similarity among the descendants. We believe that our approach is more appropriate in single domain schemas with large tree depth, which requires matching of the structural context of descendant nodes in a robust manner.

S-Match[43] is a hybrid matcher which carries out semantic matching by using the WordNet dictionary. It tackles the matching as a propositional satisfiability problem. It demonstrates better mappings than COMA++ and CUPID[66] but has worse

performance. S-Match employs 16 (13 element level and 3 structural level) match algorithms. It does not fulfill the requirement for merging and creating mappings from the source to the integrated schema. PORSCHE specifically addresses the data integration needs of large environments where efficiency is important.

Mork and Bernstein [78] present a case study of matching two large ontologies of human anatomy. They use lexical and hierarchical matching modules from CUPID[66] and a structural algorithm called Similarity Flooding[75] to find mappings. The hierarchical algorithm goes one step further than COMA++. The similarity of descendants is used to evaluate ancestor similarity, and the approach is not limited just to parent-child relationships. The authors argue that the hierarchical approach produced disappointing results because of differences in context. They report that a lot of customisation was required to get satisfactory results. Our approach, in comparison, considers only top-down similarity, and looks for matching ancestors only. It seems that further research is needed to properly define the role of node context in XML, in particular the scope of context, in terms of node distance and structure within a tree.

QOM[32] is a semi-automatic (RDF based) ontology mapping tool. It uses heuristics and ontological structures to classify candidate mappings as promising or less promising. It uses multiple iterations, where in each iteration the number of possible candidate mappings is reduced. It employs label string similarity (sorted label list) in the first iteration, and, afterwards, it focuses on mapping change propagation. The structural algorithm follows top down (level-wise) element similarity, which reduces time complexity. PORSCHE matching also uses label similarity using linguistic rather than lexical algorithms. In QOM, in the second iteration, depth-first search is used to select the appropriate mappings from among the candidate mappings.

Another interesting schema matching domain under active research is matching across query interfaces of structured web databases. Web page layout forms a hierarchy backed by a database schema. For certain Web domains, such as travel, these interfaces are very numerous. [50, 96] handle holistically the integration of these structured layouts as a mining problem. He and colleagues [50] observe that Web database query interfaces in the same domain are usually semantically similar, as a label is often unambiguous in a domain but it can have several meanings in a dictionary, and synonymous labels are rarely co-present in the same schema. However, grouping of elements such as *LastName* and *FirstName* under the same parent is common, as those together form a larger concept. He et al.[50] utilise data mining techniques on the input forms and data ranges, for elements available from the web

pages. The technique is more biased towards the accuracy of integration than performance. Thus, it is ideal for scenarios where the schemas are small, as in web query/ data interfaces. Our method targets tree schema structures (or schemas which can be converted to trees) which come with minimum information (just element labels) and each schema can have thousands of elements. This gives PORSCHE a much broader application domain.

Clustering, both at the element and schema level, is often used in matching and merging. Smiljanic and co-authors[94] suggest element clustering within a schema, in a repository of schemas which are matched to a given person schema. The technique shows how the combinatorial nature of the matching problem can be reduced to polynomial, and, further, to linear complexity, by using element clustering. The method only caters for schema matching. Lee et al. [60] present an integration method based on the clustering of similar DTDs of XML sources. The method works incrementally, by creating clusters of similar schemas. Similar schema selection is based on element similarity. The solution uses external oracles defining element similarity. It mainly focuses on the integration process but lacks the mediation aspect.

Instance level schema matching tools work on a sample mapping set. They use data and example sets to learn a strategy to compute equal instances and concepts, and are characterised by a high time complexity. For example, GLUE[27] uses Relaxation Labelling to calculate schema mappings: mapping assigned to an entity is typically influenced by the features of the node's neighbourhood in the graph. Multiple iterations have to be performed to confirm a mapping. Overall, such methods are slow. An advantage of instance based learners is their capability to learn mapping expressions, as illustrated in iMAP[22], a variant implementation of GLUE.

5.5 Lessons Learned

Implementing PORSCHE gave us a good insight into the existing schemas and their properties. We learned that newer standard schemas available on the web can be used together with linguistic techniques, and that string similarity should not be the sole criterion for label matching. Element names are now becoming more structured and meaningful, and new matching techniques are increasingly based on external oracles like WordNet. Linguistic label similarity can help in minimizing the target search space for matching very significantly. In a large scale scenario, performing clustering just once will enhance performance. The application of tree mining further reduces the time to select the correct target for mapping.

For schema integration purposes, the selection of one of the schemas as the seed for the initial mediated schema follows no fixed rule. We tried random, the smallest and the largest schema from the input set of schemas as the initial selection. However, using such heuristics showed that the selection depending on schema size does not boost performance. Rather, it is the structure of the initial seed schema which influences performance: if the initial mediated schema contained more frequent sub-trees shared with the set of input schema trees, the speed of matching improved.

Overall, the architecture of PORSCHE is flexible, and can accommodate new syntactic and linguistic similarity algorithms. Most importantly, PORSCHE is scalable, as demonstrated, and can be used in large-scale data integration. At present, it uses a limited array of linguistic methods but its domain specific matching quality is approximately equivalent to other current tools.

In the future, we will investigate the application of this technique in information systems based on P2P architectures. Secondly, we want to enhance the linguistic matching part of the system. Our study of the tree mining technique reveals that it can be utilised to identify relationships between the elements and groups of elements within a single tree and in a forest of schema trees. This will help in identifying subsumptions and overlaps for $n : m$ complex mappings[83]. Another possible extension is the development of persistent indexes for incremental matching.

5.6 Conclusion

We have presented a novel schema integration method, PORSCHE, which has shown very promising results for large scale schema integration. It uses a tree based depth-first traversal algorithm for matching, merging and mapping a set of schema trees. To improve performance, we adapted a technique from tree mining used in the clustering of similar node labels. This minimises the target search space for a node match and improves performance. PORSCHE uses an optimistic top down depth-first match traversal (parents are mapped before children, and the left sub-tree is traversed before the right sub-tree), since our assumption is that we utilise it in a domain specific environment. This helps in using the structural contextual semantics of nodes for better quality matching.

The originality of our method is fourfold. First, we support automated schema matching. Second, we not only generate matches, but also build an integrated schema at the same time. Third, our approach scales to hundreds of schemas. Fourth, the use of tree mining techniques for schema matching is also new in this

field. Next two chapters are devoted to our approach for complex matching. First the idea of mini-taxanomies is presented along with their uses. And following that we show how these mini-taxonomies can be used to verify complex matchings between two schemas.

Chapter 6

Applying Tree-Mining for Domain Taxonomy Discovery

Today, ontologies are being used to model a domain of knowledge in the semantic web [1]. OWL is considered to be the main language for developing such ontologies. It is based on the XML model, which inherently follows the hierarchical structure. In this chapter we demonstrate an automatic approach for emergent semantics modeling of ontologies. We follow the collaborative ontology construction method without the direct interaction of domain users, engineers or developers. A very important characteristic of an ontology is its hierarchical structure of concepts. We consider large sets of domain specific hierarchical structures as trees and apply frequent subtree mining for extracting common hierarchical patterns. The evaluation of the technique shows that these hierarchical patterns are good enough to represent and describe the concepts for the domain ontology. We utilise these patterns as mini-taxanomies to validate complex matches. The method is described in next chapter regarding complex match discovery. The technique further demonstrates the construction of the taxonomy of domain ontology. In this regard we consider the largest frequent tree or a tree created by merging the set of largest frequent sub-trees as the taxonomy. We argue in favour of the trustabilty for such a taxonomy and related concepts, since these have been extracted from the structures being used within the specified domain.

6.1 Ontology Engineering Overview

Discussion on ontology building and utilisation has been around since early 90s. Ontology has been defined in [45] as an explicit, formal specification of a shared conceptualisation of a domain of interest. Formalization aspect highlights the ma-

chine readability of the ontology and shared conceptualization points toward its acceptance by the users of the domain. Initial ontology development endeavors resulted in the form of DAML[1] and OIL [2] languages. Today the features of the two languages have been extended to OWL[3], based on XML model.

Initial focus in ontology design has been the manual technique but with the passage of time more and more semi-automatic techniques have emerged, facilitated by ontology editing tools[4]. The semi-automatic approach called ontology learning [15].

Ontology learning is a combination of tasks organised as a layered approach, in the manner of increasing complexity. The tasks are enumerated by Paul Buitelaar et al. in [15] as term extraction, synonym and translation detection, concept formulation, concept hierarchies, relations, rule derivation and axiomatization.

Concept hierarchy, also called taxonomy (is-a relation), is a tree structure of classifications for a given set of ontological objects. It is considered to be the ontology backbone. At the top of this structure is a single classification, the root node, that applies to all objects. Nodes below this root are more specific classifications that apply to subsets of the total set of classified objects. So, for instance, in common schemes of books, the root is called "Book", followed by nodes for the type: Art, Science, Fiction, Sports, etc. And each instance of "Book" concept can have properties like author, title, publisher etc. (Figure 6.1)

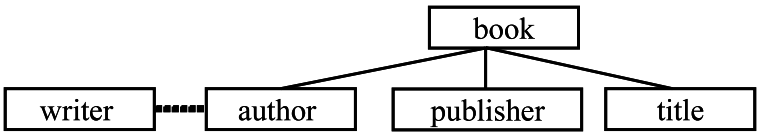

Figure 6.1: Ontology taxonomy example.

Our work is a step toward automatic conceptualisation of an ontology for a certain domain, already populated with a large set of user defined hierarchical meta data structures for diverse applications. For example, XML schemas, taxonomies or entities from which hierarchical structures can be extracted, like web based query interface forms, can be used.

[1]DAML: Darpa agent Markup Language - http://www.daml.org/
[2]OIL: Ontology Interface Layer - http://www.ontoknowledge.org/oil/
[3]OWL: Web Ontology Language - http://www.w3.org/TR/owl-features/
[4]Protege is a free, open source ontology editor and knowledge-base framework; http://protege.stanford.edu/

6.2 Related Work

One of the foremost techniques applied for ontology learning has been *term extraction* from text. Similar terms are clustered together for further analysis of inter term relations or taxonomy. These methods have their roots in natural language processing research [16]. Buitelaar et al. present their OntoLT approach as a plug-in for Protégé ontology editor. The authors define preconditions using XPATH expressions over the XML based linguistic annotations. The rules help in constructing or extending an ontology. The preconditions revolve around the linguistic constructs in a sentence, for example, the subject in the sentence corresponding to a certain morphological stem of a word.

Term similarity computation has been researched in two ways, primarily by using readily available lexical resources like Wordnet[5], and secondly, by devising clustering algorithms based on the syntactic similarity of the terms. Information retrieval techniques [92] based on term indexing and data mining methods [50] provide the space for such algorithms.

There is no definite definition available for *concept formation*. Our approach follows the hierarchical representation of concepts [33] which can be extended, upon receiving further information about the concept. The extension idea has been analysed in [69] as binary relation extraction of terms and recommendations have been made for use of data mining co-occurrence algorithms. These methods can ultimately provide an incremental approach for ontology learning.

World wide web has also been extensively exploited in this regard. [101] describe a tool which analyses the web resources like Wikipedia, Wiktitionary, along with domain corpus for domain ontology learning. These resources are exploited against a set of candidates extracted from a set of ontology instances using the linguistic context. Another work by Maedche et al. [67] details two algorithms, top-down and bottom-up approaches, for deducing taxonomic relations from the web based on heuristics. Our approach presents a similar top-down method, by applying tree-mining on the available hierarchical structures in a domain. In [44], the authors present the use of semi-structured schemata (XML and RDF based resources) for constructing a domain ontology, manually and semi-automatically.

Another interesting research for ontology generation is the use of tables extracted from the web and other resources. Authors in [97] argue that the extraction of relational knowledge from tables is much easier than exploiting the text corpus. The research describes a comprehensive framework for assembling human created tables.

[5] http://wordnet.princeton.edu

The approach canonicalises each table information, generates a mini-ontology from it and then incrementally merges the mini-ontologies.

Social collaborative networks present a new range of emerging semantics on the web. In such environments users set up lightweight conceptual structures, assigning arbitrary keywords, called tags, to resources. Such conceptual structures are also called folksonomies. Research work in [55] presents a data mining technique for discovering shared conceptualisations in folksonomies. The technique extends the data mining task of discovering all closed itemsets to frequent tri-concepts (user, tag and resource) extraction. With social networks gaining more ground, standards are also evolving for them. FOAF [6](Friend of a Friend) ontology standard is one such example, providing a structural data model for the folksonomies and paving the way for tree mining techniques in the social network environments.

6.3 Our Approach: ExSTax

In this section, we present our approach, ExSTax (**Ex**traction of **S**tructurally Coherent Mini-**Tax**onomies), for detection of ontological concepts as mini-taxonomies, from the available domain specific hierarchical structures. We discuss the architecture, and the related definitions and methods in length to clarify the novelty of our method.

We utilize the concepts of Schema Tree (Definition 1.1), Label Semantics (Definition 5.1) and Node Scope (Definition 5.3) and node scope properties of Descendant (Property 3), Ancestor (Property 5) and Right-hand-side node (Property 6) as given in section 5.1.

6.3.1 Architecture

The architecture of our approach for ontology taxonomy learning through tree mining is shown in Figure 6.2. The approach is composed of five modules: (i) *Pre-Phase*, (ii) *Similar Terms Computation and Clustering*, (ii) *Concepts Formulation*, (iv) *Similar Mini-Taxonomies Generation* and (v) *Trustable Base Taxonomy Construction*, supported by a repository which houses oracles and concept taxonomies.

The system is fed a set of hierarchical structures (schema trees). *Pre-Phase* module processes the input as trees, calculating the depth-first node number and scope (Definition 5.3) for each of the nodes in the input schema trees. At the same time, for each tree a listing of nodes is constructed, sorted in depth-first traversal

[6]http://xmlns.com/foaf/0.1/

order. As the trees are being processed, a sorted global list of distinct node labels, over the whole set of input, is created. The process is similar to pre-mapping part of PORSCHE architecture described in section 5.2.1.

In *Similar Terms Computation and Clustering* module, similarity is derived for the tree nodes labels of the input trees. We tokenise the labels and expand the abbreviated tokens using an abbreviation oracle. Currently, we utilise a domain specific user defined abbreviation table. Further, token similarity is supported by a manually defined domain specific synonym table. Label comparison is based on similar token sets or similar synonym token sets (described in section 5.2.1 as label conceptualisation). The architecture is flexible enough to employ additional abbre-

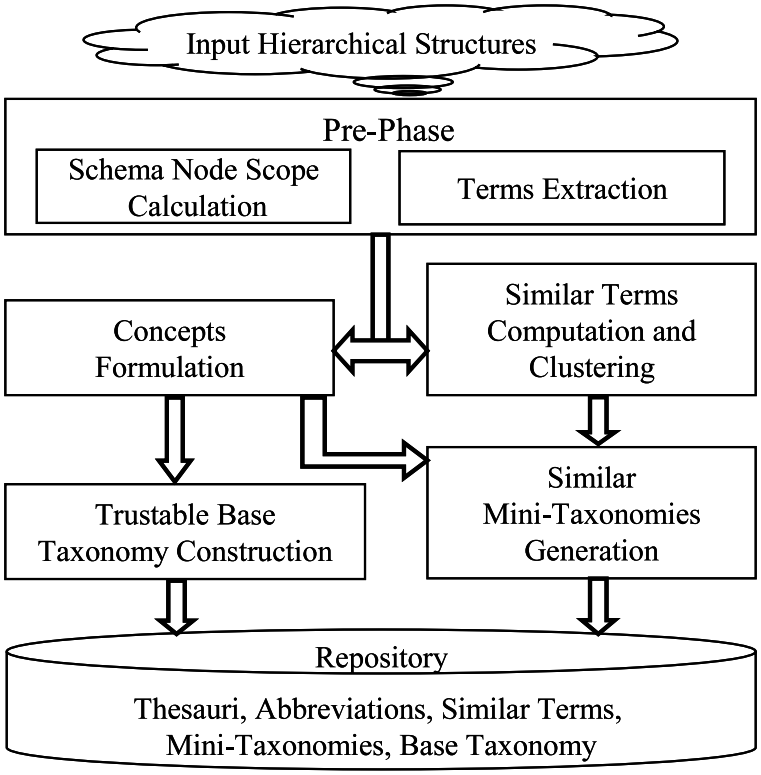

Figure 6.2: Architecture for the tree mining of ontology concepts and taxonomy.

viation, synonym oracles or arbitrary string matching algorithms. To further refine the similarity, we employ the structural aspect. Label instances at nodes in different trees are compared for ancestor level label instance similarity (Property 2). Any such occurrence helps in re-enforcing the similarity of the current pair of labels and removes any ambiguity. Based on similarity, the terms are clustered together.

In our approach, a concept is considered to be a small tree structure, referred to as a mini-taxonomy. *Concepts Formulation* module discovers these mini-taxonomies. We utilise an extended version of the frequent sub-tree mining approach described

in [104] for this purpose. Once the set of mini-taxonomies has been extracted, the set is fed to the *Similar Mini-Taxonomies Generation* module. At this stage all possible similar mini-taxonomies are generated with the help of already computed similar label clusters. The set of largest possible frequent sub-trees, from the output of concepts formulation module, acts as the input of *Trusted Base Taxonomy Construction* module. If there is just one tree, it is considered as the base taxonomy, else all the sub-trees in the set are merged together to produce the base taxonomy.

The *Repository* is an indispensable part of the system. It houses oracles: thesauri and abbreviation lists. It also stores extracted terms, inter-term similarity, mini-taxonomies representing concepts and trustable base taxonomy. It also provides persistent support to the taxonomy learning process.

6.3.2 ExSTax Algorithm and Data Structures

The algorithm implementing the *Concept Formulation* process acts as the kernel of our approach. It iterates, by extracting growing frequent sub-trees from a given set of trees. The sub-tree frequency support in the forest of trees is a user defined parameter. The algorithm takes as input the list of labels, with similar labels linked together to form a cluster (each cluster can have one or more labels). First task performed by the algorithm is to compute the frequency of each label in the forest of trees (Figure 6.4a). Next, within each cluster, the label with the highest frequency in the forest of input trees is taken as the symbol representing the cluster. The frequency of the cluster symbol is computed by adding frequencies of all the labels in the cluster. Logically, all node labels in a cluster are replaced by the cluster symbol in the input set of trees (Figure 6.4b).

We consider symbol as the representation of a sub-tree. For example, the symbol for a sub-tree with one node is the node label, and the symbol representing the tree S_1 in Figure 6.3 is "*book-author-name//publisher-name//title*" (- and / delimiters are used to signify the downward and upward traversal within the tree, respectively).

From here on the process executes similar to frequent sub-tree mining algorithm given in [104]. In the first iteration, the process finds frequent sub-trees with size 1 (Figure 6.4b), and creates the vertical list data structure for further joining, referred to as join-list. Join-list entry is a composition of three elements; (i) tree number in which the sub-tree occur, (ii) the node numbers sequence representing the sub-tree which is the prefix of the rightmost node in the sub-tree, and (iii) scope of the right most node in the sub-tree. Only sub-trees with frequency equivalent or greater than the threshold are kept in the list. Threshold frequency is computed as *support multiplied by number of the input schemas divided by hundred.*

In the second pass, a new list of join-lists is created. Each frequent size 1 sub-tree is joined with every other size 1 sub-tree in the first join-list. The joining process first evaluates the similarity of element (i) and (ii) of the sub-tree join-list entries. If the pair passes the similarity test, it is subjected to Property 2 test. If the pair passes the Property 2 i.e., descendant test is true for the pair, a new symbol for the sub-tree of size 2 is created. If the sub-tree symbol does not exist in the second list, it is added to the list. The join-list entry of the symbol is added to its respective list. Likewise, subsequent size 2 sub-trees are added to the list. At the end of this iteration, frequency of each sub-tree is computed and only sub-trees with equivalent or higher frequency than threshold are kept in the list. The iterative process keeps executing till the sub-tree list does not have any frequent sub-trees. For joining sub-trees of size 2 or greater, Property 3 (cousin test) is also evaluated for computing a prospective candidate sub-tree symbol.

The last list of sub-trees contains either one or more sub-trees. This list acts as the input for computing the base taxonomy for the given set of hierarchical structures.

6.4 A Mini-Taxonomies Extraction Example

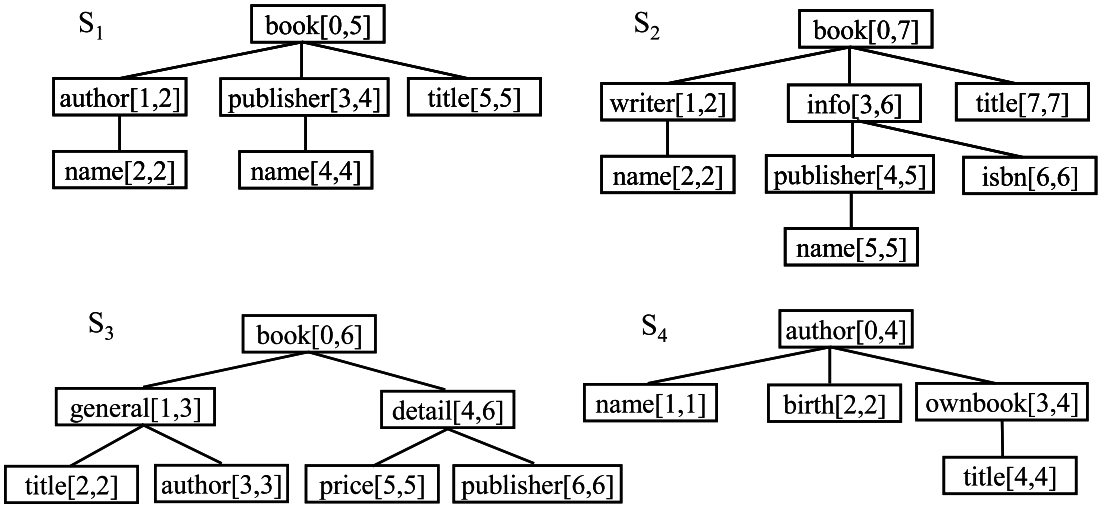

Figure 6.3: Input set of 4 trees for learning base taxonomy using tree mining.

Figure 6.3 shows four trees after Pre-Phase. A list of labels created in this traversal is enumerated in Figure 6.4a with the similar labels clusters. Incremental execution of ExSTax algorithm is demonstrated in Figure 6.4b. There are six iterations before the algorithm stops, when it is not possible to generate any larger frequent sub-trees. The sub-tree generated in the last iteration can be considered as

a. Symbols List with frequency and
Similar clusters (author, writer), (book, ownbook) and (detail, info)

author	book	birth	detail	general	info	isbn	name	ownbook	price	publisher	title	writer
2	3	1	1	1	1	1	5	1	1	3	4	2

b. Symbols representing sub-tree of size 1-6 with frequency greater than threshold and
vertical join lists {tree number, prefix sub-tree, [number, scope (of right most node)]}

author	book	detail	name	publisher	title
1,,[1,2]	1,,[0,5]	2,,[3,6]	1,,[2,2]	1,,[3,4]	1,,[5,5]
2,,[1,2]	2,,[0,7]	3,,[4,6]	1,,[4,4]	2,,[4,5]	2,,[7,7]
3,,[3,3]	3,,[0,6]		2,,[2,2]	3,,[6,6]	3,,[2,2]
4,,[0,4]	4,,[3,4]		2,,[5,5]		4,,[4,4]
			4,,[1,1]		

* - indicates downwards move and / upwards move in the tree structure

author-name	book-author	book-detail	book-name	book-pub	book-title	detail-pub	pub-name
1,1,[2,2]	1,0,[1,2]	2,0,[3,6]	1,0,[2,2]	1,0,[3,4]	1,0,[5,5]	2,3,[4,5]	1,3,[4,4]
2,1,[2,2]	2,0,[1,2]	3,0,[4,6]	1,0,[4,4]	2,0,[4,5]	2,0,[7,7]	3,4,[6,6]	2,4,[5,5]
4,0,[1,1]	3,0,[3,3]		2,0,[2,2]	3,0,[6,6]	3,0,[2,2]		
			2,0,[5,5]		4,3,[4,4]		

book-author/detail, book-author-name, book-author/name, book-author/pub, book-author/title, book-detail/pub, book-name/pub, book/name/title, book-pub-name, book-pub/title

book-author/detail-pub, book-author-name//name, book-author-name//pub, book-author-name//title, book-author/name/title, book-author/pub-name, book-author/pub/title, book-name/pub/title, book-pub-name//title

book-author-name//name/title, book-author-name//pub-name, book-author-name//pub/title, book-author/pub-name//title

book-author-name//pub-name//title
1,01234,[5,5] 2,01245,[7,7]

Figure 6.4: List of frequent sub-trees symbols, size 1 to 6 with 50% support in the input trees.

the base taxonomy for the given set of hierarchical structures. Figure 6.5 illustrates the taxonomy structure generated for the scenario.

The six iterations are presented in the six panels of Figure 6.4b. First iteration takes into account sub-trees of size one. Prefix sub-tree indicates the sub-tree to which the current node is the right most node. In first iteration the prefix data structure is empty, since there is no prefix sub-tree. Each sub-tree symbol's (e.g. author, book, author-name, book-author/detail) vertical list entry is paired with other symbols' vertical list entries. The joining of vertical lists results in a structure of size two i.e., one sub-tree can only be descendant of the other in this case. The sub-trees which are present in at least two of the input trees (50% support), are added to the second list. In vertical list entry, last number in prefix entry denotes

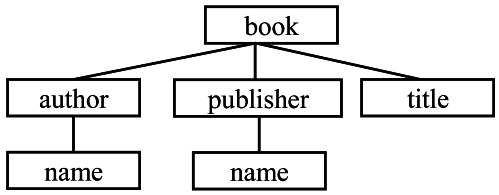

Figure 6.5: Trusted extracted taxonomy.

the number of the right most node of the prefix sub-tree (Figure 6.4b).

In subsequent iterations, both descendant test (Property 1) and cousin test (Property 3) are applied to come up with frequent sub-trees. Consider the creation of mini-taxonomy "*book-author/pub*" from the second iteration in the example by joining subtrees of size 2. Let symbol "*book-author*" list, list A, be joined to symbol "*book-pub*" list, list B. List A, entity *1,0,[1,2]* is joinable to list B entity *1,0,[3,4]*, since the schema and prefix elements of the two entities are similar. Property 1, descendant test, is not true for the two entities but the cousin test is true i.e., the right most node scopes are not overlapping (Property 3). Similarly for list A, entity *3,0,[3,3]* is joinable to list B entity *3,0,[6,6]* and it also passes the Property 3. This supports the 50 percent threshold frequency, and implies that the sub-tree with symbol "*book-author/pub*" is a frequent sub-tree of size 3.

Panels 3-5 present the symbols of extracted frequent sub-trees (mini-taxonomies) of sizes 3 to 5. The last panel of Figure 6.4b gives sub-tree with symbol composed of six labels. There are two vertical list elements, with the 50% support condition. The sub-tree (Figure 6.5) is present in input structures 1 and 2 (Figure 6.3).

6.5 Evaluation

The prototype implementation uses Java 5.0. A PC with Intel Xeon, 2.33 GHz processor and 2 GB RAM, running Windows XP was used. We have selected two data sets[7], BOOKS (synthetic) and COURSES (real) as the input structures for our experiments.

We examined the semantic quality of generated mini-taxonomies using the precision measure. Our target was to generate semantically meaningful taxonomic structures. Therefore, we manually scrutinized the generated tree patterns and computed the share of semantically applicable sub-trees among all found. With reference to Figure 4 structure S_1, a sub-tree structure "*book[0,5]-name[2,2]/name[4,4]*" is considered to be invalid, since it is semantically meaningless. Based on these consider-

[7]http://www.lirmm.fr/PORSCHE/TaxonomyLearning/

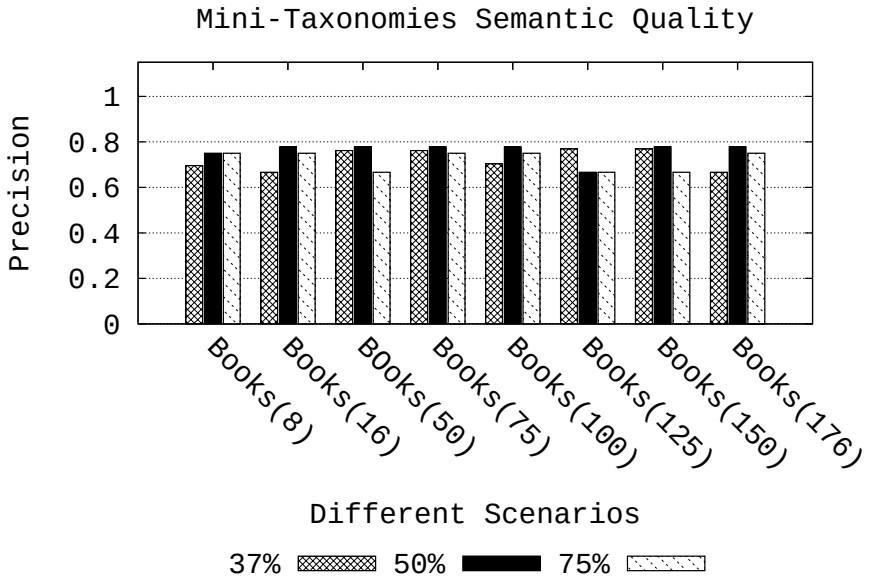

Figure 6.6: Precision of ExSTax for eight sets of tree structures from Books domain.

ations we show the precision measure computed from the experiments. Figure 6.6 shows the precision of 8 sets of input structures comprising of 8, 16, 50, 75, 100, 125, 150 and 176 trees taken from BOOKS. The results are computed for three different tree mining support values, 37, 50 and 75 percent.

In the other experiment performed on the COURSES domain of XML schema instances, with support value set to 25 percent, we retrieved with precision nearly equal to 1. The base taxonomy generated in this experiment was Course-Title/Instructor/Room/Time.

Discussion

The experimental results show the precision measure for Books domain to be between 0.65 and 0.8 and support the validity of our idea of mini-taxonomies extraction. The number of mini-taxonomies generated increased with decrease in the value of the tree mining support parameter and vice versa. Therefore, we selected the support values range (37-75), where results could be verified manually. Secondly, it is quite difficult to estimate the recall measure in the experiments because of the large number of possible outputs. Devising a system for this purpose is out of the scope of current

Table 6.1: Characteristics of schema trees used in the experiments.

Domain	BOOKS	COURSES
Number of Schemas	176	42
Average nodes per schema	8	8
Largest schema size	14	17
Smallest schema size	5	2
Schema Tree Depth	3	4

work. Further, we would tend to research the finding of valid patterns missing from the generated set, to estimate the recall measure. Another observation made is that ExSTax algorithm shows exponential scalability with respect to the size of input tree structures. Since we are concerned with the semantic validity of the output, we have not taken into account the time complexity of the algorithm.

6.6 Conclusion

We have introduced a novel cross-discipilinary technique based on tree mining, for ontology taxonomy learning. The core idea behind this work is to demonstrate the applicability of tree mining techniques for ontology taxonomy extraction in a large scale scenario. The technique inherently supports the collaborative ontology learning by holistically exploiting the already available hierarchical structures in the domain.

We have investigated its scalability with respect to the number of schemas. The experimental results demonstrate that our approach scales to hundreds of schemas. The linguistic matching of node labels uses tokenisation, abbreviations and synonyms. Our method provides an almost automated solution to the large scale domain specific taxonomy learning problem.

In next chapter we evaluate the advantages of the automatically extracted mini-taxonomies for the discovery of n:m complex mappings in the context of research described in [33].

Chapter 7

Complex Match Discovery

Match cardinality aspect in schema matching is categorized into *simple element level matching* and *complex structural level matching*. Simple matching comprises of 1:1, 1:n and n:1 match cardinality, whereas n:m match cardinality is considered to be complex. Most of the existing approaches and tools give good 1:1 local and global match cardinality but lack the capabilities for handling the complex cardinality issue. In this chapter we demonstrate an automatic approach for the creation and validation of n:m schema mappings. Our technique is applicable to hierarchical structures like XML Schema. Basic idea is to propose an n:m node mapping between children (leaf nodes) of two matching non-leaf nodes of two schemas. The similarity computation of the two non-leaf nodes is based upon the syntactic and linguistic similarity of node labels supported by similarity among the ancestral paths from nodes to the root. The n:m mapping proposition is then verified with the help of mini-taxonomies extracted from a large set of same domain schema trees. The mini-taxonomies are automatically extracted using frequent sub-tree mining approach: the higher the frequency, the higher the confidence of reliability[1]. The verification algorithm performs a comparison between the mini-taxonomies and the subtrees rooted at non-leaf nodes, which assures the authenticity of proposed n:m mapping. We support our approach with the help of quality evaluation experiments. We also present arguments to support our approach for large scale semantic web usage.

7.1 Related Work

Simple matching with quality has been successfully demonstrated in [3, 43, 74] by utilising element level and structural level schema knowledge. There has been very limited work on complex schema mapping. Use of data instance and machine

[1]ExSTax - Method for automatic extraction of mini-taxonomies described in chapter 6

learning techniques for simple and complex schema matching has been shown in [27, 22] for relational and XML schemas. [100] presents a user-centric approach using fixed point iteration algorithm similarity flooding (SF) [74] and other linguistic techniques to calculate complex mappings.

Authors in [22] demonstrate a semi-automatic tool called iMAP, which discovers 1:1 and 1:n mappings between two relational schemas. The approach utilises machine learning algorithms on data instances to predict 1:n mapping expressions. It employs knowledge like past complex matches, domain integrity constraints, and overlap data. In [50], He et al. describe a mining technique for integrating a large number of web query interfaces, considered as schemas. The proposed framework holistically exploits the set of input schemas, thus exploring the contextual information beyond two schemas. The algorithms DCM and MGC make it feasible to compute the n:m schema matching. The iMAP and DCM/MGC approaches exploit schemas with low tree depth (schema considered as hierarchical structures) and show a very limited use of structural matchers.

Another interesting work regarding XML document transformation [58] presents comprehensively the 1:1, 1:n and n:1 schema matching between two schemas. The algorithms used in this work are massively dependent on schema structural information. The proposed method worked in two steps. First, preliminary matchings between leaf nodes of the two schemas are computed using an external domain ontology and leaf node similarity. In the second step the contextual aspect of nodes is computed as path similarity, i.e., similarity between the ancestor nodes of the current nodes being matched. In the same application domain, work by Boukottaya and Vnoirbeek [14] further enhances the contextual aspect of the nodes. The approach takes into account three levels of context similarity: ancestor, child and leaf. The two research works help in exploiting the hierarchical structures with larger depth.

Our approach as given in [89], presents a large scale schema integration framework, which uses a top down matching approach. The technique follows the idea that some level of ancestor similarity is a must for an element down the hierarchy. Linguistic and tree mining algorithms are used to compute element level matchings. Likewise, QOM [32] is a robust, time performant method for RDF based schema and ontology matching. Use of ontologies/mini-ontologies for finding complex matchings has also been demonstrated in some works. Embley et al. in [33] use manually created mini-ontologies for domain specific concepts, to generate n:m correspondences between two schemas.

7.2 Complex Matching - Our Approach

In this section we describe our approach toward complex match discovery. We base our work on the concepts of Hierarchical Structure/Schema as tree, Label Semantics and Node Scope and scope properties (Ancestor and Right-hand-side node) as given in section 5.1. Specifically, we use the following scope properties.

Node Scope - Binary Properties for x [X,Y], $x_a[X_a, Y_a]$, and $x_r[X_r, Y_r]$:
• **Property (Ancestor (x, x_a), x_a is an ancestor of x)**: $X_a <$X and $Y_a \geq$Y.
• **Property (RightHandSideNode (x, x_r), x_r is Right Hand Side Node of x with Non-Overlapping Scope)**: $X_r > Y$.

7.2.1 Architecture

The architecture of our approach for complex match discovery is shown in Figure 7.1. The approach is composed of five modules: (i) *Pre-Phase*, (ii) *Similar Terms Computation and Clustering*, (iii) *Mini-Taxonomies Generation*, (iv) *Simple Schema Matching*, and (v) *Complex Match Proposition Validation*, supported by a repository which houses oracles and concepts' mini-taxonomies, schemas and match results for future reuse.

First, the system is provided a set of XML schema instances. *Pre-Phase* module processes the input as trees, calculating the depth-first node number and scope (Figure 7.4) for each of the nodes in the input schema trees. At the same time, for each schema tree a listing of nodes is constructed, sorted in depth-first traversal order. As the trees are being processed, a sorted global list of labels over the whole set of schemas is created by the *Terms Extraction* sub-module.

In *Similar Terms Computation and Clustering* module, label concepts are computed using linguistic techniques. We tokenise the labels and expand the abbreviated tokens using an abbreviation oracle. Currently, we utilise a domain specific user defined abbreviation table. Further, we make use of token similarity, supported by an abbreviation table and a manually defined domain specific synonym table. Label comparison is based on similar token sets or similar synonym token sets. The architecture is flexible enough to employ additional abbreviation or synonym oracles or arbitrary string matching algorithms. Similar labels are clustered together and each cluster is represented by a label from the respective cluster, which is the most frequent label within the set of input schemas.

Mini-Taxonomies Generation module uses a tree mining approach to extract frequent sub-trees from the set of schema trees. We use frequent pattern growth

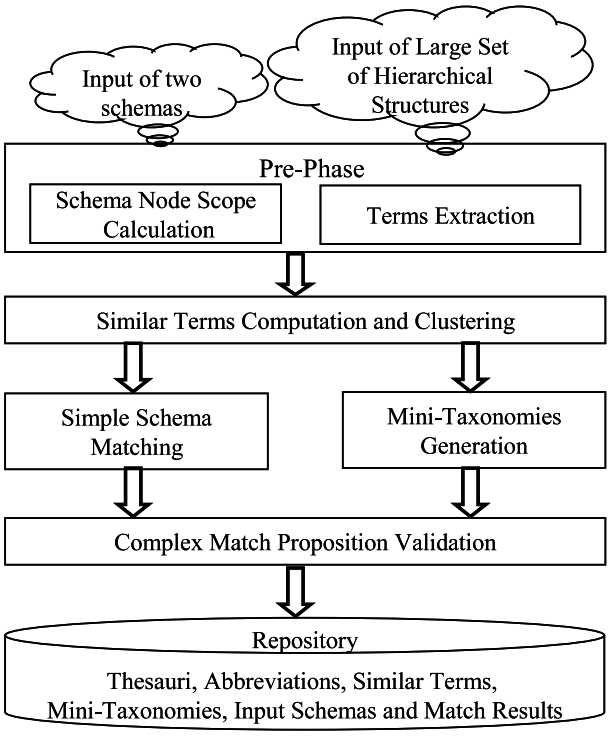

Figure 7.1: Architecture for complex match discovery using automatically generated mini-taxonomies.

mining algorithm based on research in [104]. During the process all labels within a cluster of similar labels are logically replaced by the cluster representative label. This helps in computing the right count of frequency for a label concept [88].

Simple Matching module, generates simple matchings based upon labels' linguistic similarity and node's contextual similarity using node scope values. The node scope criterion is used to evaluate the ancestor similarity factor with good time performance. We utilise almost the same tree mining technique for matching as described in chapter 5 for PORSCHE. Complex match proposition validation (CMPV) module forms the main core of this research work. We use a novel algorithm to validate the proposed complex matches with the help of automatically generated domain specific mini-taxonomies. We provide a detailed discussion about it in the following sections.

The *Repository* is an indispensable part of the system. It houses oracles: thesauri and abbreviation lists. It also stores schemas and mappings, and provides persistent support to the mapping discovery process.

```
Algorithm : nodeScope
   Data: p, c, V, d
   p: parent node list element, c: current node,
   V: nodes list, d: node depth
   Result: V
1  begin
2  |  x ← New nodesListElement(c)
3  |  x.number ← length(V)
4  |  x.parentNode ← p
5  |  x.rightMostNode ← ∅
6  |  x.depth ← d
7  |  Add x to V
8  |  if c has no children then
9  |  |  update(x, x)
10 |  d ← d + 1
11 |  for each child of c do
12 |  |  nodeScope(x, child, V, d)
13 end
```

Figure 7.2: Algorithm for schema tree node scope calculation, also computing node depth.

7.2.2 Simple Match Discovery

In this section we describe the *Simple Schema Matching* module. Before we move on, first we give a brief description of Pre-Phase module. The Pre-Phase module is similar to the Pre-Mapping module of PORSCHE, discussed in chapter 5. The basic difference lies in the computation of the node depth value for each node in a schema tree. The depth feature is introduced in the *nodeScope* algorithm as given in figure 7.2. Parameter d gives the depth level within the tree during the depth-first traversal. We utilise the node depth value for adjusting the match confidence related to ancestor mapping. At the start of the recursive algorithm for node scope calculation, value of d is 0, reflecting level 0 or root node level. d is incremented at L7.2.10, before the recursive call for next level (L7.2.12)[2]. *Terms Extraction* method executes similar to *Label Conceptualisation* module of PORSCHE.

Simple Schema Matching module is a customized form of the *Node Mapping* module of PORSCHE method. In this case we only match two schemas, looking for possible mappings based upon the linguistic and contextual (ancestor) similarities. The method suggests mappings with cardinality 1:1, 1:n, n:1 or n:m. Overall, the data structures utilised in simple schema matching are similar to the ones discussed in section 5.2.2.

The proposed n:m complex matchings are then validated with the help of mini-

[2]Lx.y refers to line y of algorithm in figure x

```
Algorithm : SimpleMatch
   Data: V1, V2 Lists of nodes of Schema 1 and 2 respectively
 1 Lx Source list V1 node label
 2 Lxtl Set of labels of candidate nodes in target list V2
 3 Vt Set of candidate nodes in target list V2
 4 xt Target node
 5 α Similarity Confidence
 6 β Match Type
 7 begin
 8     for each node x ∈ V1 do
 9         Lx ← lab(x)
10         Lxtl ← similarLabels(Lx, V2)
11         if Lxtl ¬ ∅ then
12             Vt ← VT(Lxtl)
13             αtemp ← 0.0
14             α ← 0.0
15             for each node xti ∈ Vt do
16                 αi ← ancestorMap(x, xti)
17                 if αi > αtemp then
18                     αtemp← αi
19                     α ← αi
20                     xt ← xti
21             if (Leaf(x)∧Leaf(xt)) then
22                 β ← 11
23             else
24                 if (¬Leaf(x)∧ ¬Leaf(xt)) then
25                     β ← nn
26                 else
27                     if (Leaf(x)∧ ¬Leaf(xt)) then
28                         β ← 1n
29                     else
30                         if (¬Leaf(x)∧ ¬Leaf(xt)) then
31                             β ← n1
32             map(x,xt,β,α)
33 end
```

Figure 7.3: Algorithm for simple matching between two schemas.

taxonomies representing possible concepts. After the validation process, the actual n:m mapping from a set of elements from source schema to a set of elements in target schema is created. New set data structures representing the n and m elements and n:m mapping representation are introduced at this point.

During simple match process, a match for every node (L7.3.8) from source schema tree list V_1 to target schema tree list V_2 is calculated. For each input node x, a set V_t of possible mappable target nodes in the target schema is created, producing the target search space for x. The criterion for the creation of this set of nodes is node label equivalence or partial equivalence (L7.3.9-10). V_t can have one or several

(L7.3.11) candidate target nodes. Next step is the ancestor map existence check. The method returns a value α based on the ancestor node mapping and the structural proximity of the mapping node to the current node. If the target search space V_t has more than one node, method *ancestorMap* is executed for each perspective matching node (L7.3.15-20). The candidate target node with the highest α value is selected as the mappable target node.

In *ancestorMap* method two functions are executed in parallel. Firstly, it checks for map existence for some ancestor node of current source node and secondly the proximity of the ancestor node from the current node. Ancestor node check is performed by utilising the ancestor node scope property, and if a map exists, then the proximity is calculated and returned as α. To proceed further, another added check is confirmed that the target schema node to which the source schema ancestor node is mapped to, is also an ancestor of the candidate target node. This is a variation of *upward cotopy distance* discussed in research [68]. The ancestor map checking also helps us partially to solve the inverted node mapping problem (discussed below).

The technique searches for an ancestor node xa of the current source node x, which has a mapping to a node ya in the target schema such that ya is an ancestor node of suggested target node y. The rule discards ancestor nodes which do not fulfil this criteria. Example 7.1 describes this advantage.

The following formula shows the ancestor map structuaral proximity formula.

$$\alpha = 1/(ddif_s + ddif_t)$$

where $ddif_s$ is the *d*epth *dif*ference between the current source node and the ancestor node for which a mapping exists and $ddif_t$ is the depth difference between the candidate target node and the ancestor node in target schema to which the source ancestor node is mapped to. The maximum value for α is 0.5, i.e., when the source schema ancestor node is the parent of current source node ($ddif_s$=1) and the target schema node to which it is mapped to is the parent node of candidate target node($ddif_t$=1).

Example 7.1 (Inverted Node Similarity): Consider the two schema trees S_1 and S_2 in figure 7.4. Top-down depth first matching traversal guides us to match S_1[0,5]book <-> S_2[0,5]book, S_1[1,5]author <-> S_2[1,5]writer (*author* is synonym of *writer*). Next, we have S_1[2,2]name <-> S_2[3,3]name with $\alpha = 0.33$ and following is the S_1[3,5]address <-> S_2[2,5]address. This overcomes the problem when *address* and *name* have a sibling relationship in source schema tree and a parent child relationship in target schema tree.•

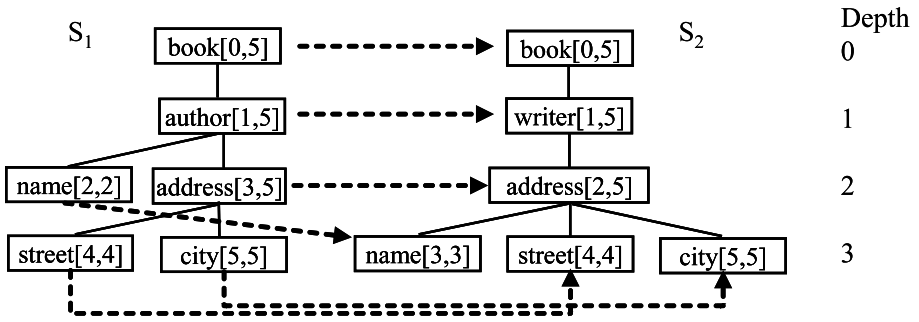

Figure 7.4: Input hierarchical structures with partial inverted node scenario.

After selecting the best match for the source node, the method proposes the type of mapping. By default it is a 1:1 mapping, since we are comparing one node from source schema to one node in target schema. Our method is more XML schema oriented where leaf nodes denote the real data values. Therefore we propose the type of mapping biased toward the leaf nodes. We extend the match type proposition given in section 5.1 with n:m match possibility.

i) 1:1 - one node of source schema corresponds to one node in the target schema; leaf:leaf or non-leaf:non-leaf.
ii) 1:n - one node in the source schema is equivalent to a composition of n leaves in the target schema; leaf:non-leaf, where a source leaf node is mapped to a set of n leaf nodes of the subtree rooted at the non-leaf node in the target schema.
iii) n:1 - n leaves in source schema compositely map to one leaf in the target schema; non-leaf:leaf, allowing a set of n leaf nodes of the subtree rooted at the non-leaf node in the source schema, to be mapped to a target leaf.
iv) n:m - n leaves in source schema compositely map to a composition of m leaves in the target schema; non-leaf:non-leaf, allowing a set of n leaf nodes of the subtree rooted at the non-leaf node in the source schema, to be mapped to a set of n leaf nodes, of subtree rooted at the non-leaf node in target schema.

Possible match type selection is handled in simple match algorithm with the help of parameter β (L3.21-31). The value of β is set to 11 for 1:1, $1n$ for 1:n, $n1$ for n:1 and nn for n:m match. Finally, the map function creates the possible mapping from source node to target node (L3.32). The n:m complex match propositions between non-leaf nodes are further exploited in the complex match proposition validation module. The algorithm is given in Figure 7.5.

7.2.3 Mini-Taxonomy Snippets Aspect

Before we move to the complex match proposition validation part, we present another data structure, *Mini-Taxonomy Snippets*, which helps in the validation process. In this section we present briefly our approach for detection of these *Mini-Taxonomy Snippets* or ontological concepts, as hierarchical structures, from available domain specific schema tree structures. A domain concept is considered to be a small tree structure which we call a Mini-Taxonomy. The idea is similar to research done in [33]. However, we follow the automatic approach rather than the manual snippet designing in [33]. The detailed explanation is given in chapter 6 [88].

The technique has an iterative nature based on incrementally extracting frequent sub-trees from a given set of trees. The sub-tree frequency in the forest of trees is a user defined parameter. The algorithm takes as input the list of terms, with similar terms linked together to form a cluster (each cluster can have one or more terms). First task performed by the algorithm is to compute the frequency of each term in the forest of trees. Next, within each cluster, the term with the highest frequency in the forest of trees is taken as the symbol representing the cluster. The frequency of the cluster symbol is computed by adding frequencies of all the terms in the cluster. From here on the algorithm executes similar to frequent growing sub-tree mining algorithm given in [104]. The similar label cluster symbols are treated as the starting labels for the data structure showing frequent sub-tree patterns. After the process is complete, we have a list of sets of mini-taxonomies. Each list represents a set of mini-taxonomies of same size. Next, we replicate this list of sets of mini-taxonomies, by replacing the cluster level similar labels in the list. This produces all possible mini-taxonomies which can be considered as concept representation, frequently utilised by domain users.

7.2.4 Complex Match Proposition Validation

The basic idea behind the CMPV (Complex Match Proposition Validation) is to validate the n:m match propositions created in the simple matching module. Our approach tries to utilise small conceptual taxonomies already generated from a large number of schemas within a specific domain, as detailed above. For simplicity, we structure these mini-taxonomies as individual XML schema instances and generate a data structure $MiniTax$ for them, similar to the input source and target schema, using the same *nodeScope* algorithm given in Figure 7.2.

Algorithm $ComplexMatchValidation$ takes as input V_1, V_2 (source and target schema node lists respectively) and $MiniTax$ (List of lists of mini-taxonomy nodes).

```
Algorithm : ComplexMatchValidation
   Data: V1, V2, MiniTax
   V1, V2 : Lists of nodes of schema 1 (source) and 2(target) respectively
   MiniTax : List of lists of mini-taxonomies
   Each mini-taxonomy list, mT,V1 and V2 sorted on depth-first order
 1 sLeaf : Set of labels of leaf nodes of non-leaf node of V1
 2 tLeaf : Set of labels of leaf nodes of non-leaf node of V2
 3 mTsLeaf, mTtLeaf : Sets of labels of leaf nodes of mini-taxonomy
 4 Fs, Ft : Boolean flags
 5 begin
 6     for each node x ∈ V1 do
 7         Fs ← false
 8         Ft ← false
 9         if (¬Leaf(x) ∧ exists(map(x,xt,β,α)) ∧ β= nn) then
10             // xt is non-leaf node in V2
11             sLeaf ← leafNodesLabels(x)
12             tLeaf ← leafNodesLabels(xt)
13             Lx ← lab(x)
14             Lxt ← lab(xt)
15             for each mT ∈ MiniTax do
16                 LmTroot ← lab(mTroot) // mTroot is root node of mT
17                 if (¬Fs ∧ LmTroot = Lx) then
18                     mTsLeaf ← leafNodesLabels(LmTroot)
19                     if sLeaf ⊂ mTsLeaf then
20                         Fs ← true
21                 if (¬Ft ∧ LmTroot = Lxt) then
22                     mTtLeaf ← leafNodesLabels(LmTroot)
23                     if tLeaf ⊂ mTtLeaf then
24                         Ft ← true
25                 if Fs ∧ Ft then
26                     break
27             if Fs ∧ Ft then
28                 β ← nn
29                 map(leafNodes(x),leafNodes(xt),β)
30 end
```

Figure 7.5: Algorithm for validation of complex matching between two schemas.

The algorithm traverses each node x of source schema, V_1, with a n:m map type (marked as nn) to xt node of target schema V_2 (L7.5.6,9). The objective of the algorithm is to certify that the leaf nodes of the sub-tree rooted at such a node x in the source schema can form an n:m mapping with the leaf nodes of sub-tree rooted at node xt in the target schema. The algorithm extracts leaf node labels from sub-tree rooted at x into a temporary set data structure $sLeaf$ and similarly populates a data structure $tLeaf$ for xt (L7.5.11,12). Next, the algorithm traverses the $MiniTax$ data structure (L7.5.15) for mini-taxonomies with root node label similar to node labels of x and xt (L7.5.17,21). Further, the leaf node labels of these mini-taxonomies are

extracted into temporary set data structures $mTsLeaf$ (root label similar to x label) and $mTtLeaf$ (root label similar to xt label) at L7.5.18,22. Finally, the validation check is executed, i.e., checking if $sLeaf$ is a subset of $mTsLeaf$ and $tLeaf$ is a subset of $mTtLeaf$. If such mini-taxonomies are found in $MiniTax$, it certifies that the concepts at x and xt are similar. Their respective leaf nodes can form a n:m mapping (L5.27-29), since the mini-taxonomies have been frequently used in the domain, which utilises the collective intelligence over a specific domain.

7.3 A Complex Match Validation Example

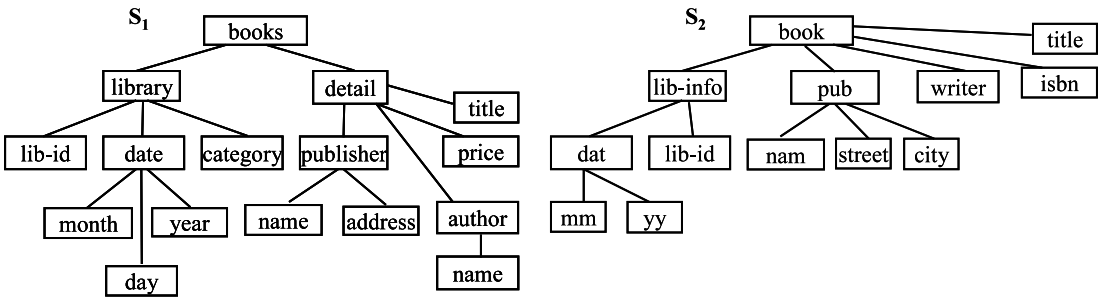

Figure 7.6: Two schema (trees) S_1 and S_2 used in complex match discovery.

Figure 7.6 shows two schema trees from the books domain. A list of correspondences is shown in Figure 7.7, after the execution of simple matching (Figure 7.3). The simple match algorithm discovers one to one matches between two corresponding elements of the two schemas. In parallel, it also proposes any possible complex match, according to the leaf/non-leaf status of the elements participating in the mapping. The possible complex match propositions are shown in brackets.

The scenario presents one n:1 and four n:m complex match situations. The n:1 match given by S_1.author[12,13]<->S_2.writer[10,10] proposes that all leaf nodes of S_1.author[12,13] compositely correspond to S_2.writer[10,10]. This establishes the fact that the data at element S_1.name[13,13] matches data represented by S_2.writer[10,10].

The analysis of the four complex matches shows that two propositions, S_1.books[0,15]<->S_2.book[0,12] and S_1.library[1,7]<->S_2.lib-info[1,5], have large sub-trees rooted at non-leaf elements, which presents a collection of concepts rather than a single concept. This fact is verified by scanning the scope of the non-leaf nodes. The possibility of such large sub-trees to be frequent in the domain is very rare.

The scrutiny of other two complex match propositions (S_1.date[3,6]<->S_2.dat[2,4], S_1.publisher[9,11]<->S_2.pub[6,9]) shows that small sub-trees reside at the elements

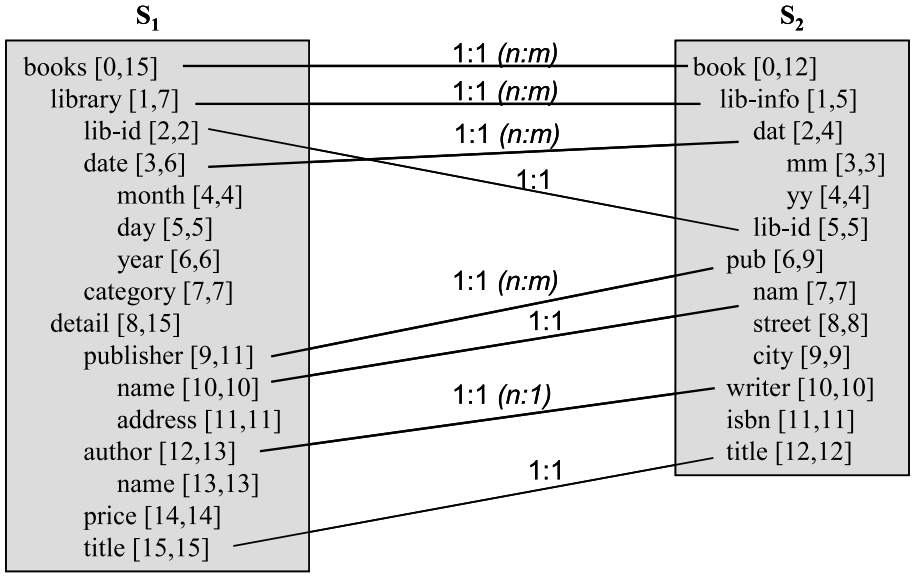

Figure 7.7: Element level mappings between schemas S_1 and S_2 after execution of simple match algorithm; complex match propositions are shown in brackets.

of interest.

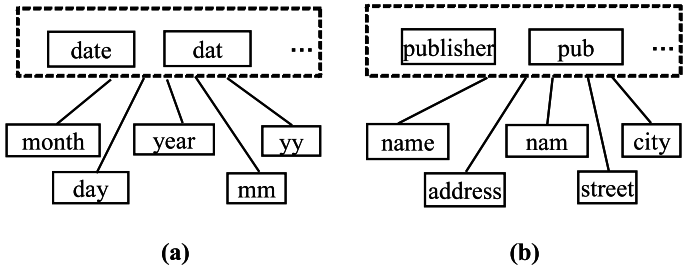

Figure 7.8: Mini-taxonomies extracted from large input of books domain schemas, using ExSTax method for complex match validation.

The validation of these proposed complex matchings is done by the Complex-MatchValidation algorithm given in Figure 7.5. For this purpose the set of already acquired mini-taxonomies (using ExSTax method) are considered. There are two mini-taxonomies, shown in Figure 7.8.(a) presents the mini-taxonomy representing the *date* concept. It can be represented by *date*, *dat* or some other similar string, as the root node for the concept. The leaf node collection of $month, day, year, mm, yy$ represent attributes describing the concept. Within one instance of the mini-taxonomy, the presnece of synonymous leaf nodes is not possible e.g., if *month* and *mm* are synonymous then the two nodes will not exist together in a mini-taxonomy with root node *date* or *dat*. (b) shows mini-taxonomies representing publisher information regarding name and address.

The execution of the validation algorithm occurs for each of the four n:m proposi-

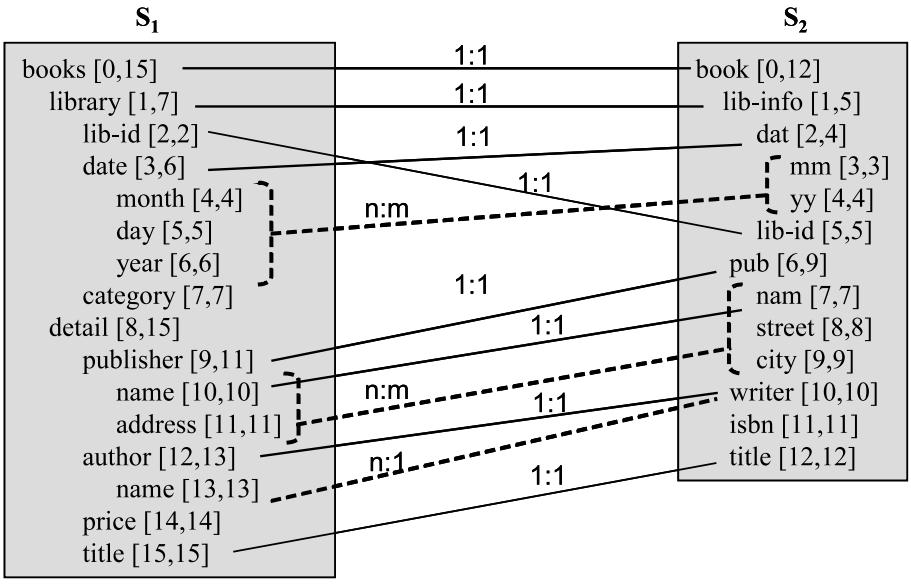

Figure 7.9: Mappings between schemas S_1 and S_2 after the execution of the complex match valdation algorithm.

tions. In case of S_1.books[0,15]<–>S_2.book[0,12] and S_1.library[1,7]<–>S_2.writer[1,5], no mini-taxonomy is found with root element similar to *books* or *book* and *library* or *lib-info*. As a result the two propositions are discarded and only the 1:1 simple match is considered. In the other two cases the algorithm finds frequent mini-taxonomies in the form of date-month/day/year, dat-mm/yy, publisher-name/address and pub-nam/street/city[3]. The algorithm substantially authenticates the propositions and creates two more complex mappings as S_1.(month[4,4],day[5,5], year[6,6])<–>S_2.(mm [3,3],yy[4,4]) and S_1.(name[10,10],address[11,11])<–>S_2.(nam[7,7], street[8,8],city[9,9]). The final mappings are shown in Figure 7.9

Table 7.1: Characteristics of domain schema trees used in the mini-taxonomy computation experiments.

Domain	BOOKS1	BOOK SEARCH	JOBS	AUTO	AIR TRAVEL	REAL ESTATE	COURSES
Type	Synthetic	Real	Real	Real	Real	Real	Real
Number of Schemas	176	19	20	14	20	14	42
Average nodes per schema	8	6	5	5	14	9	8
Largest schema size	14	12	8	10	21	20	17
Smallest schema size	5	3	4	3	6	4	2
Schema Tree Depth	3	2	2	2	2	2	4
n:m match propositions	2	2	1	1	3	2	2

[3]- and / delimeters denote downward and upward traversal within the tree, respectively

7.4 Evaluation

The prototype implementation uses Java 5.0. A PC with Intel Xeon, 2.33 GHz processor and 2 GB RAM, running Windows XP was used. We selected several data sets, BOOKS1, BOOKSEARCH [4], JOBS, AUTO, AIRTRAVEL, REALESTATE and COURSES as the input for our experiments. Characteristics of these sets of schemas in given in Tab. 7.1.

A manual analysis of the real domain schemas showed that only very small number of sets of elements (sets of leaf nodes representing some concept) could participate in a complex mapping. Since such sets are rare, finding them as frequent mini-taxonomies is even more scarce. Following is a possible list of concepts for which n:m mappings may exist in the above schemas.

(i) date/depart/return : month,day,year <–> mm,yy <–> month,day,time

(ii) address/location : name,address <–> nam,street,city <–> street1, street2,city <–> address1,address2 <–> AreaCode,Country <–> city,state

(iii) telephone : tel_res,tel_off <–> morn_tel, even_tel,night_tel <–> tel_mobile, tel_fix

(iv) name : firstName,lastName <–> f_name,mi,l_name

(v) passengers : adult,child,infant <–> adult,senior,child <–> adult_12-65,adult_65,child_2-4, child_5-12,infant

(vi) return/depart : month,day <–> month,day,year,time

(vii) schedule : days,time,room <–> DayTime,room <–>Times, Place <–> Time-Begin,TimeEnd,Room,Building <–> time,building

(viii) car model : vfrom,vto <–> fyear,tyear

To verify our CMPV method, we added some of these sets of elements to several real schemas, to extract them as mini-taxonomies. Our implementation showed the expected results, since the overall test scenarios have been synthetically generated.

Discussion

The idea of using mini-taxonomies works well, if their leaf nodes can represent some real complex matches with a contextual map at ancestor level. And the ancestor level mapping is highly dependent on label matching. Working with real world

[4] http://metaquerier.cs.uiuc.edu/repository

schemas showed (i) the possibility of a complex match is very limited, and (ii) the label matching framework should be exceptionally good.

Considering the first issue of complex match possibility, the manual scrutiny of above mentioned schemas showed a very small number of n:m matches. For example, in 20 schemas of *AIRTRAVEL* domain (with average of nodes per schema equal to 14), there were three concepts, *passengers*, *return* and *depart*, which could be considered for a complex match. But the second issue of good label matching framework deficiency gives poor quality match results. For example, there were 13 different labels for passenger concept in the domain, i.e., How many travelers are going, Number of passengers, NumAdults, adultsChildren, persons, adultnoChildno, passengers, number of travelers, how many people are going, num-pax, travelers, who is going on this trip, passengers-adults-num. Computing a high frequency of *passengers* concept (mini-taxonomies) existence was infeasible because of the multi-word, cryptic labels also including homonyms.

Our idea has some similarity to Embley et al. [33] work, which uses manually created concept ontology snippets for complex match discovery. Our approach tries to show that matching can be done automatically in a scenario using a large number of domain schemas. Secondly using the ancestor level contextual map, the simple match results overcome the inverted parent-child match problem. This makes the system more robust and dependable. Although the approach is similar to upward cotopy distance measure, it is more flexible in execution. We do a top-down traversal during the match process, i.e. parents are matched before children. Therefore, the nearest ancestor with an adequate match in the target schema is enough to verify the context.

7.5 Conclusion

In this chapter, we have presented an automatic approach for discovering a complex match. It is based on a tree mining technique supporting large size schema matching. It extends the tree mining data structure proposed in [104] and exploites ancestor scope properties (integer logical operations) on schema nodes to enable fast calculation of contextual (hierarchical) similarity between them. Its originality lies in the use of mini-taxonomies in complex match discovery. Mini-taxonomies are frequent subtrees within this forest of schema trees. Furthermore, our approach is implemented as a prototype and validated by experiments using different real schemas.

Firstly, the method computes simple element level matchings between two schemas.

The algorithm linguistically matches node labels and uses ancestor level match existence (in ontology matching called upward cotopy distance) for this purpose. At this point, the matches have 1:1 cardinality, but our method also proposes the possible complex matches intrinsically. Our approach is based on the leaf or non-leaf status of the node, putting forward the match proposition that when a non-leaf node is matched to a non-leaf node, there is the probability of an n:m match between the leaf nodes of the two non-leaf nodes. The algorithm CMPV certifies this proposition, indirectly utilising the collective intelligence of the domain users. This is achieved by using mini-taxonomy snippets, extracted from a large number of input schemas developed by users and used over the specific domain.

The work presented in this chapter has lots of prospective applications. Our approach can provide an automatic matching environment, supported by the collective intelligence of the domain users. In the future, we plan to extend the label level matching techniques. We intend to utilize state of the art lexical matchers and dictionaries. Secondly, we will be extending the idea of evolving schema matchings in the large scale scenario. Finally, our aim is to fully incorporate our technique for web based interface form (containing hierarchical nested fields) matching and collaboration among different metadata based virtual social environments.

Next chapter summarizes this dissertation and provides the insights for further extending our techniques and their applications.

Chapter 8

Conclusion and Future Research Perspectives

This is the concluding chapter of our dissertation presenting three aspects of our research work. Firstly, we give a summary of our contributions in the large scale schema matching and integration scenario. Secondly, we enumerate the limitations and related issues not covered by our proposed systems. Finally, we conclude by highlighting the future research directions in the field of schema matching and integration.

8.1 Main Contributions

In this section, we give a summary of our contributions in the field of large scale schema matching and integration.

8.1.1 State of the Art

In this thesis we have presented a detailed account of the state of the art in schema matching and integration from the large scale perspective. Chapter 3 gave the reasons why schema matching is a must module for any data intensive application. The problem's multi-dimensional aspects have been highlighted in detail as the basic schema matching functions, the related research domains like schema integration, and the application domain of this research. We also described the possible match strategies, using cross disciplinary techniques like mining and clustering, to handle the large scale scenario. Further, a discussion on methods for enhancing the match results is given in chapter 3. Finally, an analysis of available tools is provided, with respect to a simple two schema match and a large set of schemas integration

scenarios.

8.1.2 Desiderata for Schema Mediation

After analysing the state of the art in schema matching and integration, we have presented the desiderata for large scale schema matching and integration in chapter 4. We have mentioned the different aspects of schema matching and integration in a large scale scenario. The quality of mappings, integrity of the integrated schema, related measures and the time performance requirements have been described in the chapter. The match cardinality issues in the schema integration scenario are also highlighted. The chapter concludes by giving a brief outlook on the viability of mining and graphical user interface techniques for the large scale match scenarios.

8.1.3 Performance Oriented Schema Mediation

In chapter 5, we presented a new robust automatic method which discovers semantic schema matches in a large set of XML schema instances, incrementally creates an integrated schema encompassing all schema trees, and defines mappings from the contributing schemas to the integrated schema. Our approach implements the main desiderata for large scale schema matching and integration as described in chapter 4. The method, PORSCHE (Performance ORiented SCHEma mediation), utilizes a holistic approach which first clusters the nodes based on linguistic label similarity. Then it applies a tree mining technique using node ranks calculated during depth-first traversal. This minimizes the target node search space and improves performance, which makes the technique suitable for large scale data sharing. This shows the applicability of clustering and tree mining aspect in matching as proposed in chapter 3, thus giving a novel dimension for handling the large scale schema matching problem.

8.1.4 Framework for Complex Matching

Complex schema matching is within itself a complete research topic. In chapter 7 we have presented an automatic framework for discovering complex matches between two schemas. The technique utilizes an extended version of PORSCHE method, to discover possible complex matches. The approach first proposes the possibility and then verifies the propositions using already available hierarchical structures called mini-taxonomies. The mini-taxonomies are concept representation within a certain knowledge domain.

8.1.5 Mini-Taxonomies and Domain Ontology Discovery

We used growing pattern tree mining for the automatic extraction of mini-taxonomies used in a complex match framework. In chapter 6, we have demonstrated an automatic approach for emergent semantics modeling of ontologies. We followed the collaborative ontology construction method without the direct interaction of domain users, engineers or developers. A very important characteristic of an ontology is its hierarchical structure of concepts. We considered large sets of domain specific hierarchical structures as trees and applied frequent sub-tree mining for extracting common hierarchical patterns. These sub-tree patterns intuitively modeled the basic domain concepts. Further extending the technique helped us generate the basic domain ontology taxonomy.

8.2 Remaining Issues

Although we tried to address the large scale schema matching and integration problem to the maximum extent, but still there are limitations, as in any type of research. In this section we summarize the limitations and the issues not handled by our proposed systems but are related to schema matching.

8.2.1 The Limitations

In broad, our approach handles schemas represented as trees, and we have implemented only the XML schema instance wrapper to verify our approach. For a proper generic implementation, we need to develop other conversion wrapper methods i.e., for relational (supported by JDBC or ODBC), object oriented databases, OWL ontologies or XDR schemas. Secondly, we have only used some specific linguistic features of tokenisation and synonym matching, although the framework is flexible enough to benefit from other string matching functions like n-gram and edit distance for better results.

Another limitation of our system is that the main PORSCHE framework does not incorporate the complex matching algorithm for integration and mediation purposes. This requires more intelligent software to select automatically between a complex map and simple matches, in a hybrid framework for the integration and mediation of a large number of schemas.

8.2.2 Open Issues

One of the main related problems is the maintenance of mappings during schema evolution. Schema evolution can either be a change in the existing schema or an addition/ removal of a schema in the input set of schemas. Firstly, schema evolution can affect the integrity of the mediated schema, and, further the mappings need to be upgraded. Another related problem in schema evolution and maintenance is data model dependency. Overall, this is a very complex problem, which is inherently a complete field of research in computer science.

In large scale matching, tools are guided by two basic parameters, match quality and time performance. Therefore, implementations in this field need to self-tune to provide a balance between the two aspects. Another related research problem is the development of correctness/completeness metrics and benchmark tools for evaluating schema matching and integration systems.

Our work highlights the trade off between time performance and map quality. It lacks the discussion between time performance and the map complexity. Although it can be deduced that one can have better time performance if simple mappings are considered. The relation between mapping complexity and overall quality of mappings is another dimension which requires more research work, depending upon the domain of application.

8.3 Future Directions

We conclude our discussion by enumerating some explicit future research concerns in the perspective of our research detailed in the dissertation.

- Extending PORSCHE framework with more linguistic/lexical matchers, along with research technique for complex match discovery for large scale schema integration purposes.
- Large scale schema matching and integration as web service.
- Large scale schema matching and integration in schema based P2P database systems.
- Application of our matching research in domain specific large scale social network environments.
- Large scale schema matching and integration using parallel computing techniques.

- Techniques for visualization of mappings in multi-schema (more than 2) integration.
- Multi-user graphical user interface to handle large scale semi-automatic schema matching and integration.
- Domain specific ontology extraction from a large set of input schemas (representing different data models) and their instances in the same knowledge domain using mining techniques. And, learning all aspects of domain ontology, comprising of term extraction, synonym and translation detection, concept formulation, concept hierarchies, relations, rule derivation and axiomatization.

List of Publications

Accepted Research Papers

Journal

- Khalid Saleem, Zohra Bellahsene and Ela Hunt. PORSCHE: Performance ORiented SCHEma mediation. *Information Systems Journal - Elsevier, (33), number 7-8, pages 637-657, 2008.*

Conference Proceedings

- Khalid Saleem and Zohra Bellahsene. Automatic Extraction of Structurally Coherent Mini-Taxonomies. *In 27th International Conference on Conceptual Modeling (ER), Springer LNCS Vol. 5231, pages 341-354, Barcelona, October 2008.*

- Khalid Saleem, Zohra Bellahsene and Ela Hunt. Performance Oriented Schema Matching. *In 18th International Conference on Database and Expert Systems Applications (DEXA), Springer LNCS Vol. 4653, pages 844-853, Regensburg, September 2007.*

Workshop Proceedings

- Khalid Saleem and Zohra Bellahsene. A scalable approach for large scale schema matching. *In Proceedings of l'échelle des techniques de découverte de correspondences (DÉCOR) Workshop co-located with 7ème journées francophones de Extraction et Gestion des Connaissances (EGC), Namur, January 2007.*

Research Papers under Review

- Khalid Saleem and Zohra Bellahsene. New Challenges in Data Integration: Large Scale Automatic Schema Matching. *Technical Report submitted to Hal-Lirmm, April 2008.*

- Khalid Saleem and Zohra Bellahsene. Complex Schema Match Proposition and Validation by Structurally Coherent Frequent Mini-Taxonomies. *Technical Report submitted to Hal-Lirmm, September 2008.*

Bibliography

[1] G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. The MIT Press, 2004.

[2] T. Asai, H. Arimura, and S. Nakano. Discovering Frequent Substructures in Large Unordered Trees. In *ICDS*, pages 47–61, 2003.

[3] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *ACM SIGMOD*, pages 906–908, 2005.

[4] D. Bachlechner, K. Siorpaes, D. Fensel, and I. Toma. Web Service Discovery - A Reality Check. Technical report, Digital Enterprise Research Institute (DERI), 2006.

[5] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparitive Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[6] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. The MOMIS Approach to Information Integration. In *ICEIS*, pages 194–198, 2001.

[7] S. Benkley, J. Fandozzi, E. Housman, and G. Woodhouse. Data ELement Tool-based Analysis (DELTA). In *MTR*, 1995.

[8] P. A. Bernstein, S. Melnik, and J. E. Churchill. Incremental Schema Matching. In *VLDB*, 2006.

[9] P. A. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-Strength Schema Matching. *ACM SIGMOD Record*, 33(4):38–43, 2004.

[10] P. Besana, D. Robertson, and M. Rovatsos. Exploiting Interaction Contexts in P2P Ontology Mapping. In *P2PKM*, 2005.

[11] G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML Schema Definitions from XML Data. In *VLDB*, 2007.

[12] A. Bilke and F. Naumann. Schema Matching using Duplicates. In *Intl. Conf. on Data Engineering*, 2005.

[13] P. Bohannon, E. Elnahrawy, W. Fan, and M. Flaster. Putting Context into Schema Matching. In *VLDB*, 2006.

[14] A. Boukottaya and C. Vanoirbeek. Schema Matching for Transforming Structured Documents. In *ACM DocEng*, 2005.

[15] P. Buitelaar, P. Cimiano, and B. Magnini. Ontology Learning from Text: An Overview. In *Ontology Learning from Text: Methods, Evaluation and Applications Frontiers*. IOS Press, 2005.

[16] P. Buitelaar, D. Olejnik, and M. Sintek. A protege plug-in for ontology extraction from text based on linguistic analysis. In *ESWS*, 2004.

[17] K. C. C. Chang, B. He, M. Patel, and Z. Zhang. Structured databases on the web: Observations and implications. *SIGMOD Record*, (3), 2004.

[18] H. Chen, Z. Wu, G. Zheng, and Y. Mao. RDF-based schema mediation for database grid. In *Fifth IEEE/ACM Workshop on Grid Computing*, 2004.

[19] L. Chiticariu, P. G. Kolaitis, and L. Popa. Interactive Generation of Integrated Schemas. In *ACM SIGMOD*, 2008.

[20] T. Dalamagasa, T. Chengb, K.-J. Winkelc, and T. Sellisa. A methodology for clustering XML documents by structure. *Information Systems*, 31:187–228, 2006.

[21] M. Davis. Semantic Wave 2006 - A Guide to Billion Dollar Markets - Keynote Address. In *STC*, 2006.

[22] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *ACM SIGMOD*, pages 383–394, 2004.

[23] H. H. Do, S. Melnik, and E. Rahm. Comparison of Schema Matching Evaluations. In *Web, Web-Services, and Database Systems Workshop*, 2002.

[24] H.-H. Do and E. Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885, 2007.

[25] A. Doan. *Learning to Map between Structured Representations of Data.* PhD thesis, University of Washington, 2002.

[26] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling Schemas of Disparate Data Sources - A Machine Learning Approach. In *ACM SIGMOD*, 2001.

[27] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Y. Halevy. Learning to match ontologies on the Semantic Web. *VLDB J.*, 12(4):303–319, 2003.

[28] F. Duchateau, Z. Bellahsene, and R. Colleta. A Flexible Approach for Planning Schema Matching Algorithms. In *COOPIS*, 2008.

[29] F. Duchateau, Z. Bellahsene, and E. Hunt. XBenchMatch: a Benchmark for XML Schema Matching Tools. In *VLDB*, pages 1318–1321, 2007.

[30] F. Duchateau, Z. Bellahsene, and M. Roche. A Context-based Measure for Discovering Approximate Semantic Matching between Schema Elements. In *IEEE RCIS*, 2007.

[31] M. Ehrig, J. Euzenat, and H. Stuckenschmidt. Framework for Ontology Alignment and Mapping - Results of the Ontology Alignment Evaluation Initiative. In *Workshop on Integrating Ontologies*, 2005.

[32] M. Ehrig and S. Staab. QOM - Quick Ontology Mapping. In *ISWC*, pages 683–697, 2004.

[33] D. W. Embley, L. Xu, and Y. Ding. Automatic Direct and Indirect Schema Mapping: Experiences and Lessons Learned. *ACM SIGMOD Record*, 33(4):14–19, 2004.

[34] J. Euzenat. Semantic Precision and Recall for Ontology Alignment Evaluation. In *Intl. Joint Conf. on Artificial Intelligence*, 2007.

[35] J. Euzenat et al. State of the art on ontology matching. Technical Report KWEB/2004/D2.2.3/v1.2, Knowledge Web, 2004.

[36] J. Fong, F. Pang, and C. Bloor. Converting Relational Database into XML Document. In *12th International Workshop on Database and Expert Systems*, 2001.

[37] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *ACM SIGMOD Record*, 34(4):27–33, 2005.

[38] A. Gal. Managing Uncertainty in Schema Matching with Top-K Schema Mappings. *JoDS*, pages 90–114, 2006.

[39] A. Gal. Why is schema matching tough and what can we do about it. *SIGMOD Record*, 35(4), 2006.

[40] A. Gal, A. Anaby-Tevor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB J*, 14(1):50–67, 2005.

[41] A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari. Sweetening WordNet with DOLCE. *AI Magzine*, 24(3):13–24, 2003.

[42] J. E. C. George G. Robertson, Mary P. Czerwinski. Visualization of Mappings Between Schemas. In *ACM CHI*, 2005.

[43] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an Algorithm and an Implementation of Semantic Matching. In *European Semantic Web Symposium*, pages 61–75, 2004.

[44] A. Gomez-Perez and D. Manzano-Macho. Deliverable 1.5: A survey of ontology learning methods and techniques. Technical report, Universidad Politecnica de Madrid, 2003.

[45] T. Gruber. Towards principles for the design of ontologies used for knowledge sharing. *Human and computer Studies Journal.*, 43:907–928, 1994.

[46] W. Guedria, Z. Bellahsene, and M. Roche. A Flexible Approach based on the User Preferences for Schema Matching. In *IEEE RCIS*, 2007.

[47] L. M. Haas. Beauty and the Beast: The Theory and Practice of Information Integration. In *Intl. Conf. on Database Theory*, 2007.

[48] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *Intl. Conf. on Data Engineering*, pages 505–516, 2003.

[49] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data Integration: The Teenage Years. In *VLDB*, 2006.

[50] B. He, K. C.-C. Chang, and J. Han. Discovering complex matchings across web query interfaces: a correlation mining approach. In *ACM KDD*, pages 148–157, 2004.

[51] H. He, W. Meng, C. T. Yu, and Z. Wu. Automatic integration of Web search interfaces with WISE-Integrator. *VLDB J.*, 13(3):256–273, 2004.

[52] M. A. Hernedez, R. J. Miller, and L. M. Haas. Clio: A Semi-Automatic Tool for Schema Mapping. In *ACM SIGMOD*, 2002.

[53] W. Hu, Y. Zhao, and Y. Qu. Partition-Based Block Matching of Large Class Hierarchies. In *Asian Semantic Web Conf.*, 2006.

[54] M. N. Huhns and M. P. Singh. Service-Oriented Computing: Key Concepts and Principals. *IEEE Internet Computing*, 2005.

[55] R. Jasche, A. Hotho, C. Schmitz, B. Ganter, and G. Stumme. Discovering shared conceptualizations in folksonomies. *Web Semantics: Science, Services and Agents on World Wide Web*, 6(1):38–53, 2008.

[56] A. Jhingran. Enterprise Information Mashups: Integrating Information, Simply - Keynote Address. In *VLDB*, pages 3–4, 2006.

[57] D. Lee, M. Mani, F. Chiu, and W. W. Chu. NeT CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints. In *ACM CIKM*, 2002.

[58] J.-S. Lee and K.-H. Lee. Computing simple and complex matchings between XML schemas for transforming XML documents. *Information and Software Technology - Elsevier*, 48:937–946, 2006.

[59] M.-L. Lee, T. W. Ling, H. Lu, and Y. T. Ko. Cleansing Data for Mining and Warehousing. In *Intl. Conf. on Database and Expert Systems Applications*, 1999.

[60] M.-L. Lee, L. H. Yang, W. Hsu, and X. Yang. XClust: clustering XML schemas for effective integration. In *ACM CIKM*, pages 292–299, 2002.

[61] Y. Lee, M. Sayyadain, A. Doan, and A. S. Rosenthal. eTuner: tuning schema matching software using synthetic scenarios. *VLDB J.*, 16:97–122, 2007.

[62] M. Lenzerini. Data Integration: A Theoretical Perspective. In *ACM PODS*, 2002.

[63] A. Loser, W. Siberski, M. Sintek, and W. Nejdl. Information Integration in Schema-Based Peer-To-Peer Networks. In *CaiSE*, 2003.

[64] J. Lu, S. Wang, and J. Wang. An Experiment on the Matching and Reuse of XML Schemas. In *Intl. Conf. on Web Engineering*, pages 273–284, 2005.

[65] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based Schema Matching. In *Intl. Conf. on Data Engineering*, pages 57–68, 2005.

[66] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *VLDB*, pages 49–58, 2001.

[67] A. Maedche, V. Pekar, and S. Staab. Ontology learning part one – on discovering taxonomic relations from the web. In *Web Intelligence*, 2002.

[68] A. Maedche and S. Staab. Measuring similarity between ontologies. In *EKAW*, 2002.

[69] A. Maedche and S. Staab. Ontology Learning. In S. Staab and R. Studer, editors, *Handbook of Ontologies*. Springer Verlag, 2004.

[70] D. Manakanatas and D. Plexousakis. A Tool for Semi-Automated Semantic Schema Mapping: Design and Implementation. In *DisWEB Workshop CaiSE*, 2006.

[71] t. . I. Maria da Conceicao Moraes Batista and Ana Carolina Salgado. In *Quality in Databases, VLDB workshop*, 2007.

[72] R. McCann, W. Shen, and A. Doan. Matching Schemas in Online Communities: A Web 2.0 Approach. In *Intl. Conf. on Data Engineering*, 2008.

[73] D.-E. Meddour, M. Mushtaq, and T. Ahmed. Open Issues in P2P Multimedia Streaming. In *MultiComm*, 2006.

[74] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128, 2002.

[75] S. Melnik, E. Rahm, and P. A. Bernstein. Developing metadata-intensive applications with Rondo. *J. of Web Semantics*, I:47–74, 2003.

[76] P. D. Meo, G. Quattrone, G. Terracina, and D. Ursino. Integration of XML Schemas at various "severity" levels. *Information Systems*, pages 397–434, 2006.

[77] P. Mitra, N. F. Noy, and A. R. Jaiswal. OMEN: A Probabilistic Ontology Mapping Tool. In *Intl Semantic Web Conf.*, pages 537–547, 2005.

[78] P. Mork and P. A. Bernstein. Adapting a Generic Match Algorithm to Align Ontologies of Human Anatomy. In *ICDE*, pages 787–790, 2004.

[79] I. Niles and A. Pease. Towards a Standard Upper Ontology. In *FOIS*, 2003.

[80] N. F. Noy, A. Doan, and A. Y. Halevy. Semantic Integration. *AI Magazine*, 26(1):7–10, 2005.

[81] N. F. Noy, S. Kunnatur, M. Klein, and M. A. Musen. Tracking Changes During Ontology Evolution. In *Intl. Semantic Web Conf.*, 2004.

[82] T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 2nd edition, 1999.

[83] T. Pankowski and E. Hunt. Data Merging in Life Science Data Integration Systems. In *Intelligent Information Systems*, pages 279–288, 2005.

[84] C. Parent and S. Spaccapietra. Database Integration: The Key to Data Interoperability. In M. P. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Advances in Object Oriented Modeling.* The MIT Press, 2000.

[85] C. Pluempitiwiriyawej and J. Hammer. Element matching across data-oriented XML sources using a multi-strategy clustering model. *DKE*, 48(3):297–333, 2003.

[86] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[87] H. Roitman and A. Gal. OntoBuilder: Fully Automatic Extraction and Consolidation of Ontologies from Web Sources using Sequence Semantics. In *EDBT Workshops*, 2006.

[88] K. Saleem and Z. Bellahsene. Automatic Extraction of Structurally Coherent Mini-Taxonomies. In *ER*, 2008.

[89] K. Saleem, Z. Bellahsene, and E. Hunt. PORSCHE: Performance ORiented SCHEma mediation. *Information Systems - Elsevier*, (7-8):637–657, 2008.

[90] A. D. Sarma, X. Dong, and A. Halevy. Bootstrapping Pay-As-You-Go Data Integration Systems. In *ACM SIGMOD*, 2008.

[91] N. Schurr, J. Marecki, M. Tambe, and P. Scerri. The Future of Disaster Response: Humans Working with Multiagent Teams using DEFACTO. In *AAAI Spring Symposium on Homeland Security*, 2005.

[92] H. Schutze. Word Space. In *NIPS*, pages 895–902, 1993.

[93] P. Shvaiko and J. Euzenat. A Survey of Schema-Based Matching Approaches. *J. Data Semantics IV*, pages 146–171, 2005.

[94] M. Smiljanic, M. van Keulen, and W. Jonker. Using Element Clustering to Increase the Efficiency of XML Schema Matching. In *Workshop Intl. Conf. on Data Engineering*, page 45, 2006.

[95] S. Spaccapietra, C. Parent, and Y. Dupont. Model Independent Assertions for Integration of Hetrogeneous Schemas. *VLDB J.*, pages 81–126, 1992.

[96] W. Su, J. Wang, and F. Lochovsky. Holistic Query Interface Matching using Parallel Schema Matching. In *Intl. Conf. on Data Engineering*, pages 122–125, 2006.

[97] Y. A. Tijerino, D. W. Embley, Y. Ding, and G. Nagy. Towards Ontology Generation from Tables. *World Wide Web*, 8:261–285, 2005.

[98] A. Tomasic, L. Raschid, and P. Valduriez. Scaling access to heterogeneous data sources with DISCO. *IEEE Trans. on Knowledge and Data Engineering*, 10(5):808–823, 1998.

[99] Y. Velegrakis, R. Miller, and L. Popa. On Preserving Mapping Consistency under Schema Changes. *VLDB J*, 13(3):274–293, 2004.

[100] G. Wang, V. Zavesov, R. Rifaieh, A. Rajasekar, J. Goguen, and M. Miller. Towards User Centric Schema Mapping Platform. In *VLDB Workshop Semantic Data and Semantic Integration*, 2007.

[101] N. Weber and P. Buitelaar. Web-based Ontology Learning with ISOLDE. In *ISWC WorkShops Web Content Mining with Human Language*, 2006.

[102] W. Wu, A. Doan, and C. Yu. Merging Interface Schemas on the Deep Web via Clustering Aggregation. In *Intl. Conf. on Data Mining*, 2005.

[103] M. Yatskevich. Preliminary Evaluation of Schema Matching Systems. Technical Report DIT-03-028, Informatica e Telecomunicazioni, University of Trento, 2003.

[104] M. J. Zaki. Efficiently Mining Frequent Embedded Unordered Trees. *Fundamenta Informaticae*, (1-2):33–52, 2005.