MANAGEMENT OF XML DATA BY MEANS OF SCHEMA MATCHING

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.),

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von: M.Sc. Alsayed Alshahat Alsayed Algergawy
geb. am 06. Nov. 1973 in Ägypten

Gutachter:

Prof. Dr. Gunter Saake

Prof. Dr. Erhard Rahm

Prof. Dr. Stefan Conrad

Promotionskolloquium: Magdeburg, Germany, February 19, 2010

Algergawy, Alsayed:
Management of XML Data by Means of Schema Matching
Dissertation, Otto-von-Guericke-Universität Magdeburg, 2009.

Abstract

The eXtensible Markup Language (XML) has emerged as a de facto standard to represent and exchange information among various applications on the Web and within organizations due to XML's inherent data self-describing capability and flexibility of organizing data. As a result, the number of available (heterogeneous) XML data is rapidly increasing, and the need for developing high-performance techniques to manage these data is vastly growing. A first step to manage these data is to identify and discover semantic correspondences across XML data. The process of identifying semantic correspondences among heterogeneous XML data is called XML schema matching.

Schema matching in general plays a central role in several shared XML data applications, such as XML data integration, XML data migration, XML data clustering, peer-to-peer systems, etc. Therefore, myriads of matching algorithms have been proposed and many matching systems have been developed. However, most of these systems produce score schema elements, which results in discovering simple (one-to-one) matches. Such results solve the schema matching problem partially. In order to completely solve the problem, the matching system should discover complex matches as well as simple ones. Another dimension of schema matching that should be considered is matching scalability. Existing matching systems rely heavily either on rule-based approaches or on learner-based approaches. Rule-based systems represent schemas to be matched in a common data model, such as schema trees or schema graphs. Then, they apply their algorithms to the common data model, which in turn requires traversing schema trees (schema graphs) many times. By contrast, learning-based systems need much pre-match effort to train their learners. As a consequence, especially in large-scale schemas and dynamic environments, matching efficiency declines radically. As an attempt to improve matching efficiency, recent schema matching systems have been developed. However, they only consider simple matching. *Therefore, discovering complex matching taking into account schema matching scalability against both a large number of schemas and large-scale schemas is considered a real challenge.*

This thesis proposes a new matching approach, called *sequence-based schema matching*, to identify and discover both simple and complex matches in the large-scale XML schema context. The approach is based on exploiting the Prüfer encoding method that constructs a one-to-one correspondence between schema trees and sequences. As a result of sequence-based schema matching we develop two approaches in sequence. To begin with, we develop the *XPrüM* framework, which identifies and discovers simple (one-to-one) matches by representing schema trees as sequences. By exploiting this representation we capture both schema tree internal (semantic) information in the *Label Prüfer Sequence (LPS)* and schema tree external (structural) information in the *Number Prüfer Sequence (NPS)*. Capturing both information in this efficient way provides and maximizes the possibility to get better matching results. To assess the internal similarity between XML schema elements, we develop a linguistic element matcher that exploits semantic information in LPSs, while to assess the structural similarity between schema elements, we propose a structure matcher that makes use of structural information in NPSs. Then, to cope with complex matches, we further enhance the *XPrüM* framework by introducing the concept of compatible elements.

We also present two case studies where our sequence-based matching approach can be deployed. Moreover, the thesis introduces a new evaluation measure, *cost-effectiveness*, to consider both performance aspects: *matching effectiveness* and *matching efficiency*.

The *XPrüM* and its enhancement frameworks as well as the two case studies have been designed, developed and implemented. The frameworks have been evaluated on various real world test cases with encouraging results, thus, empirically proving their benefits.

Zusammenfassung

Die eXtensible Markup Language (XML) hat sich durch ihre inhärente Eigenschaft der Selbstbeschreibung von Daten und die Flexibilität bei der Organisation von Daten zum Industriestandard zur Darstellung und zum Austausch von Informationen zwischen verschiedenen Anwendungen im Web und in Organisationen entwickelt. Als Ergebnis wächst die Menge verfügbarer (heterogener) XML-Daten rapide an, und die Notwendigkeit, hochperformante Techniken zur Verwaltung dieser Daten zu entwickeln, steigt erheblich. Ein erster Schritt, um diese Daten zu verwalten, ist die Identifikation und Entdeckung semantischer Korrespondenzen innerhalb der XML-Daten. Der Prozess der Identifikation semantischer Korrespondenzen zwischen heterogenen XML-Daten wird als XML Schema Matching (dt. Schemaabgleich) bezeichnet.

Allgemein hat Schema Matching eine zentrale Bedeutung für verschiedene Anwendungen gemeinsam genutzter XML-Daten, wie zum Beispiel bei der Integration, der Migration oder dem Clustering von XML-Daten, in Peer-to-Peer-Systemen usw. Deshalb sind eine Vielzahl von Matching-Algorithmen und -Systemen entwickelt worden. Jedoch produzieren die meisten dieser Systeme Bewertungen für Schemaelemente, was nur zur Entdeckung einfacher (1:1) Abbildungen führt. Solche Ergebnisse lösen das Problem aber nur teilweise. Um das Problem vollständig zu lösen, sollte ein Matching-System komplexe und einfache Abbildungen entdecken. Eine weitere Dimension des Schema Matching, welche berücksichtigt werden muss, ist die Skalierbarkeit. Existierende Systeme verlassen sich entweder stark auf regelbasierte oder auf lernbasierte Ansätze. Regelbasierte Systeme repräsentieren abzubildende Schemata in einem gemeinsamen Datenmodell, zum Beispiel Schemabäume oder Schemagraphen. Anschliessend führen sie ihre Algorithmen auf dem gemeinsamen Datenmodell aus, welches ein mehrfaches Durchlaufen der Schemabäume (Schemagraphen) erfordert. Im Gegensatz dazu benötigen Systeme, welche auf Lernverfahren basieren, umfangreichen Aufwand zum Training der Learns. Als Konsequenz daraus verschlechtert sich insbesondere für grosse Schemata und in dynamischen Umgebungen die Effizienz des Abgleichs radikal. Neuere Matching-Systeme setzen sich deswegen zum ziel, die Matching-Effizienz zu steigern. Aber auch diese betrachten nur einfache Abbildungen zwischen Schemaelementen. Deshalb stellt die Entdeckung komplexer Abbildungen bei gleichzeitiger Berücksichtigung der Skalierbarkeit sowohl bezüglich einer grossen Anzahl von Schemata als auch grosser Schemata eine wirkliche Herausforderung dar.

Diese Arbeit schlägt einen neuartigen Matching-Ansatz vor, welcher als sequenzbasiertes Schema Matching bezeichnet wird und einfache und komplexe Abbildungen im Kontext grosser XML-Schemata identifiziert und entdeckt. Der Ansatz basiert auf der Verwendung von Prüfer-Codes, welche eine 1:1-Korrespondenz zwischen Schemabäumen und Sequenzen konstruieren. Für die Umsetzung des sequenzbasierten Matching entwickeln wir zwei aufeinander aufbauende Ansätze. Zuerst entwickeln wir das XPrüm-Framework, welches einfache (1:1) Abbildungen durch die Darstellung von Schemabäumen als Sequenzen identifiziert und entdeckt. Wir verwenden diese Darstellung für interne (semantische) Informationen in der Label Prüfer Sequence (LPS, Folge der Knotenbezeichnungen) und externe (strukturelle) Informationen in der Number Prüfer Sequence (NPS, eigentlicher Prüfer-Code) beider Schemabäume. Diese effiziente Darstellung beider Informationen ermöglicht und maximiert die Wahrscheinlichkeit, bessere Matching-Ergebnisse zu erhalten. Um die innere Ähnlichkeit zwischen XML-Schemaelementen zu berechnen, entwickeln wir einen linguistischen Element-Matcher, welcher semantische Informationen der zwei LPS nutzt, während zur Berechnung der strukturellen Ähnlichkeit ein Matcher vorgestellt wird, welcher Strukturinformationen der NPS nutzt. Darauf aufbauend erweitern wir das XPrüm-Framework durch die Einführung des Konzeptes der kompatiblen Elemente, um komplexe Abbildungen behandeln zu können.

Wir stellen ebenfalls zwei Fallstudien vor, in denen unser Matching-Verfahren angewandt werden kann. Darüber hinaus führt die Arbeit mit der Kosteneffektivität ein neues Evaluationsmass ein, welches beide Performanzaspekte berücksichtigt: die Effektivität und die Effizienz des Matching.

Sowohl das erweiterte Framework XPrüM als auch die beiden Fallstudien wurden entworfen, entwickelt und implementiert. Die Frameworks wurden anhand verschiedener Realwelt-Testdatensätze mit ermutigenden Ergebnissen evaluiert, und dadurch wurde ihr Nutzen empirisch nachgewiesen.

Acknowledgments

In the name of Allah (GOD), Most Gracious, Most Merciful. It is my firm belief that this dissertation has never been completed without the help of GOD. My belief in God gave me hope during difficult times. Thanks to Allah.

I would like to express my deep gratitude to everyone who helped me shape the ideas explored in this dissertation, either by giving technical advice or encouraging and supporting my work in many other ways. This dissertation would not have come into existence without their hands-on advice and motivation.

First of all, I am deeply indebted to my country (EGYPT, THE BIG MOTHER) for accepting and supporting me to do my Ph.D. in Germany. I also have to thank my small family, my mother, my wife, and my kids (Alaa and Aan) for moral support, encouragement, and understanding.

I am extremely grateful to Professor *Gunter Saake*, my scientific advisor, for guiding my work from the very first day and for supporting me in several directions. He gave me the opportunity to conduct this doctoral research and helped me made the right strategic decisions at many forks along the way. He kept me on track while allowing me to broaden my research horizon in tangential areas. His insightful comments, which densely filled the margins of each draft that I gave to him, gave rise to many creative ideas.

I am very grateful to Professor *Zohra Bellahsene*, University of Montpellier II. She gave me the first feedback and impression about the proposed approaches in this dissertation. She showed me the direction of how to improve the quality of my work. I would also like to express my thanks to Dr. *Eike Schallehn*, University of Magdeburg. He has been the driving force behind the emerging research area of schema matching, the subject of the dissertation. I am appreciative to Dr. *Richi Nayak*, Queensland University of Technology, Australia. She provided me with her inspiring guidance, remarkable suggestions, constructive criticism and friendly discussions that enabled me to complete the research work and this thesis efficiently. Dr. Nayak spared a lot of her precious time in advising and helping me throughout the research work.

I am grateful to external thesis reviewers, Prof. Dr. Erhard Rahm Leipzig University and Stefan Conrad Düsseldorf University for the time and energy they have spent in reviewing my thesis and their detailed technical feedback.

I am deeply indebted to Professor *Mahmoud M. Fahmy*, my M.Sc. supervisor. He taught me the craft of research, and learned me how to correctly write a technical writing. I also extend my sincere gratitude to all the staff of the Faculty of Engineering, Tanta University, Egypt, where I did my undergraduate and master studies.

Finally, I would like to thank all members of the Database Group at the University of Magdeburg, for their support and many informal discussions, which helped me put my academic research into perspective. A special word of thanks goes to Ms. *Stefanie Quade* for proofreading and improving the quality of the dissertation. Thanks to all my friends and colleagues here in Magdeburg/Germany, and there in Egypt.

....

Magdeburg, Germany

March 1, 2010

Alsayed Algergawy

Contents

List of Figures

List of Tables

Part I

Introduction

1 Motivations & Objectives

This chapter shortly introduces the context of this thesis, which contributes to the field of management of XML data, especially to the task of identifying and discovering semantic correspondences across XML data. In particular, we briefly describe the problems that occur during the identification of semantically similar elements among heterogeneous (XML) data sources, giving motivation for our work. We then introduce the contributions of the thesis, and finally provide an outline of the thesis.

1.1 Motivations

Schema matching is the task of identifying semantic correspondences among elements across different data sources. It plays a central role in many data application scenarios [96]: in *data integration* to identify and characterize inter-schema relationships across multiple (heterogeneous) schemas; in *data warehousing* to map data sources to a warehouse schema; in *E-business* to help to map messages between different XML formats; in the *Semantic Web* to establish semantic correspondences between concepts of different ontologies [76]; in *data migration* to migrate legacy data from multiple sources into a new one [58]; and in *XML data clustering* to determine semantic similarities between XML data [106].

At the core of most of these data application scenarios, the eXtensible Markup Language (XML) has emerged as a standard for information representation, analysis, and exchange on the Web. Since XML provides data description features that are similar to those of advanced data models, XML is today supported either as native data model or on top of a conventional data model by several database management systems. As a result, XML databases on the Web are proliferating, and efforts to develop good information integration technology for the growing number of XML data sources have become vital. Identifying and discovering semantic correspondences among heterogeneous data sources is the biggest obstacle for developing such an integrated schema. The process of identifying these correspondences across XML schemas is called *XML schema matching* [48].

As a result, myriad of matching algorithms have been proposed and many systems for automatic schema matching have been developed [117, 63]. However, most of these systems such as Cupid [96], Similarity Flooding (SF) [101], COMA/COMA++ [47, 48], LSD [49], BTreeMatch [61], OntoBuilder [68], S-Match [70] and PORSCHE [122] produce scores schema elements, which results in discovering only simple (one-to-one) matching. Such results solve the schema matching problem partially. In order to completely solve the problem, the matching system should discover complex matchings as well as simple ones. Few work has addressed the problem of discovering complex matching [44, 77, 81], because of the greater complexity of finding complex matches than of discovering simple ones.

Additionally, existing schema matching systems rely heavily either on *rule-based* approaches or on *learning-based* approaches. Rule-based systems [96, 47, 48, 70] represent schemas in a common data model, such as schema trees or schema graphs. Then, they apply their algorithms to the common data model, which in turn requires traversing schema trees (schema graphs) many times. By contrast, learning-based systems [49, 53] need much pre-match effort to train their learners. As a consequence, especially in large-scale schemas and dynamic environments, matching efficiency declines radically. As an attempt to improve matching efficiency, recent schema matching systems have been developed [127, 122]. However, they consider only simple matching. *Therefore, discovering complex matching taking into account schema matching scalability against both a large number of schemas and large-scale schemas is considered a real challenge.*

Considering matching scalability is accompanied by another schema matching challenge: *matching evaluation*. Many real-world problems, such as the schema matching problem, involve multiple measures of performance, which should be optimized simultaneously. Optimal performance according to one objective, if such an optimum exists, often implies unacceptably low performance in one or more of the other objective dimensions, creating the need for a compromise to be reached. In the schema matching problem, the performance of a matching system involves multiple aspects, among them matching effectiveness and matching efficiency. Optimizing one aspect, for example effectiveness, will affect the other aspects, such as efficiency. Hence, we need a compromise between them, and we could consider the trade-off between effectiveness and efficiency matching result as a multi-objective problem.

To summarize, the work presented in this thesis has been motivated by the following main challenges:

- introducing XML schema matching to the large-scale context;
- identifying and discovering complex matches;
- schema matching evaluation in the large-scale context.

1.2 Objectives & contributions

To face the mentioned challenges, we aim at proposing a new schema matching paradigm, called sequence-based schema matching. Our approach is based on exploiting the Prüfer encoding method that constructs a one-to-one correspondence between XML schema trees and sequences. In particular, we develop and implement an XML schema matching system, called *XPrüM*. The main objective of *XPrüM* is to cope with schema matching in the large-scale context. To deal with complex matches, the system is enhanced with introducing the concept of *compatible elements*. We deploy our matching system in several application domains, such as XML schema clustering and Web service discovery to ensure the validity and applicability of the system. We also introduce the concept of *cost-effectiveness* in order to trade-off between matching effectiveness and matching efficiency.

During the realization of our objectives, we have achieved the following contributions:

- presenting an overview of a number of shared-data applications from different domains that take advantage of schema matching;
- introducing a detailed up-to-date survey of state-of-the-art schema matching approaches and systems under a generic framework;
- introducing XML schema element similarity measures that can be used to assess the similarity between schema elements;
- designing and developing a new approach to schema matching based on the Prüfer encoding method, called sequence-based matching approach;

- designing and developing a new matching framework, called *XPrüM* to discover simple matches in the context of large-scale schemas;
- improving and enhancing *XPrüM* to cope with complex matches;
- proposing and developing algorithms for a sequence-based matching approach;
- introducing a new measure to evaluate both matching performance aspects (matching effectiveness and matching efficiency), called *cost-effectiveness*;
- conducting an intensive set of experiments to validate the proposed approach utilizing different scenarios;
- deploying our matching approaches in different application domains, XML schema clustering and Web service discovery;
- introducing an overview of future trends in the schema matching field.

Part of the material of the thesis has been published in various conferences and journals (in order of appearance). Whenever results of any of these works are reported, proper citations are made in the body of the thesis.

- [8]: A. Algergawy, E. Schallehn, and G. Saake. A unified schema matching framework. In 19. GI-Workshop on Foundations of Databases, pages 58-62. Bretten, Germany, May 2007.
- [9]: A. Algergawy, E. Schallehn, and G. Saake. Combining effectiveness and efficiency for schema matching evaluation. In First International Workshop Workshop on Model-Based Software and Data Integration (MBSDI 2008), volume 8 of CCIS, Springer, pages 19-30. Berlin, Germany, April 2008.
- [10]: A. Algergawy, E. Schallehn, and G. Saake. Fuzzy constraint-based schema matching formulation. In Business Information Systems (BIS 2008) Workshops, pages 141-152, Innsbruck, Austria, May 2008. CEUR Workshop Proceedings 333.
- [12]: A. Algergawy, E. Schallehn, and G. Saake. A Prüfer sequence-based approach for schema matching. In Eighth International Baltic Conference on Databases and Information Systems (BalticDB&IS2008), pages 205-216. Estonia, June 2008.

- [14]: A. Algergawy, E. Schallehn, and G. Saake. A sequence-based ontology matching approach. In Proceedings of the Fourth International Workshop on Contexts and Ontologies (C&O) Collocated with the 18th European Conference on Artificial Intelligence (ECAI-2008), pages 26-30. Patras, Greece, July 2008.

- [13]: A. Algergawy, E. Schallehn, and G. Saake. A schema matching-based approach to XML schema clustering. In The Tenth International Conference on Information Integration and Web-based Applications Services (iiWAS 2008), pages 131-136. Linz, Austria, ACM, Nov. 2008.

- [11]: A. Algergawy, E. Schallehn, and G. Saake. Fuzzy constraint-based schema matching formulation. Scalable Computing: Practice and Experience, Special Issue: The Web on the Move, 9(4):303-314, Dec. 2008.

- [15]: A. Algergawy, E. Schallehn, and G. Saake. Databases and Information Systems V - Selected Papers from the Eighth International Baltic Conference, volume 187, chapter A New XML Schema Matching Approach Using Prüfer Sequences, pages 217-228. ISO Press, 2009.

- [7]: A. Algergawy and G. Saake. A classification scheme for XML data clustering techniques. In 4th International Conference on Intelligent Computing and Information Systems (ICICIS 2009), pages 550-555. Cairo, Egypt, March 2009.

- [16]: A. Algergawy, E. Schallehn, and G. Saake. Efficiently locating web services using a sequence-based schema matching approach. In 11th International Conference on Enterprise Information Systems (ICEIS 2009). pages 287-290, Milan, Italy, May 2009.

- [17]: A. Algergawy, E. Schallehn, and G. Saake. Improving XML schema matching performance using Prüfer sequences. Data & Knowledge Engineering, 68(8):728-747, August 2009.

- [6]: A. Algergawy, R. Nayak, E. Schallehn, and G. Saake. Supporting web service discovery by assessing web service similarity. In 13th East-European Conference on Advances in Databases and Information Systems (ADBIS-2009). Riga, Latvia, Sept. 2009.

- [5]: A. Algergawy, R. Nayak, and G. Saake. XML schema element similarity measures: A schema matching context. In 8th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE 2009) at OTM Conferences, Vilamoura, Algarve, Portugal, LNCS 5871, pp. 1246-1253, Nov 02-04, 2009.

1.3 Road map of the thesis

The thesis is structured in five parts as follows:

Part I. Part one is devoted to mention motivation and objectives of the thesis and it presents a definition of schema matching and its environment. Chapter 1 declares thesis motivation, objectives and contributions to the schema matching field. In Chapter 2, the schema matching problem is (in)formally defined by introducing a definition for the generic matching problem and casting this definition to the schema matching problem. From this definition, we state the matching process input and output focusing on different data models that represent input schemas, and give more attention to XML schemas identifying what different heterogeneities are, what issues in representing large-scale schemas are and why they increase the complexity of schema matching.

Part II. Part two presents a comprehensive overview of schema matching state-of-the-art. Chapter 3 introduces a number of shared-data applications from different domains that can take the advantage of schema matching. Chapter 4 presents a generic schema matching framework that is used to survey a large number of existing schema matching systems and prototypes in light of generic framework phases.

Part III. Part three is dedicated to presenting our sequence-based schema matching approach. Chapter 5 introduces a number of XML schema element similarity measures that can be used to assess the similarity between XML schema elements. Chapter 6 gives the development and implementation details of frameworks that realize our approach. Chapter 7 reports the evaluation criteria for schema matching approaches as well as the settings in which we ran our experiments. It also reports the results of the conducted experiments.

Part IV. Part four is devoted to point out how the sequence-based matching approach can be deployed in different application domains. Chapter 8 introduces SeqXClust that deploys our approach in clustering XML schemas. Chapter 8 presents a web service discovery framework.

Part V. Finally, part five concludes the thesis. Chapter 9 summarizes the work done in the thesis. Chapter 10 outlines future trends in the schema matching field.

2 Preliminaries

The rapid increase of information and communication technologies has made accessible large amount of information stored in different application-specific databases and web sites. The number of different information sources is rapidly increasing and the problem of the ability of two or more information systems to exchange information and to use the information that has been exchanged is becoming more and more sever. Schema matching plays a central role in solving such problems.

Schema matching is the process of identifying semantic correspondences among elements across different data sources. However, a first step in finding high-performance techniques to solve difficult problems, such as the schema matching problem, is to build a complete, possibly formal, problem specification. A suitable and precise definition of schema matching is essential for investigating approaches to solve it. Hence, to understand and handle the complexity of the schema matching problem and to be able to advise an efficient algorithm to cope with the matching problem, a (formal) problem specification is required. This chapter is devoted to present schema matching and its environment. In particular, we first present a motivation scenario to illustrate the schema matching problem. Then, we give a definition of schema matching in order to recognize schema matching surrounds.

The material presented in this chapter has been developed and published in [8, 10].

```xml
<Schema name="Schema S"
xmlns="urn:schemas-microsoft-com:xml-data ">
  <ElementType name="AccountOwner">
    <element type="Name"/>
    <element type="Address"/>
    <element type="Birthdate"/>
  </ElementType>
  <ElementType name="Address">
    <element type="street"/>
    <element type="city"/>
    <element type="state"/>
    <element type="ZIP"/>
  </ElementType>
</Schema>
```

```xml
<Schema name="Schema T"
xmlns="urn:schemas-microsoft-com:xml-data ">
  <ElementType name="Customer">
    <element type="FName"/>
    <element type="LName"/>
    <element type="CAddress"/>
  </ElementType>
  <ElementType name="CAddress">
    <element type="street"/>
    <element type="city"/>
    <element type="province"/>
    <element type="code"/>
  </ElementType>
</Schema>
```

Figure 2.1: Two XML schemas

2.1 Motivation example

To describe the methodologies presented in this chapter, we use a real-life scenario that happens in e-business. This example has been used in [147]. To keep the chapter self-contained, we briefly present the example in this section.

Example 2.1 *For a multinational company, there are two subsidiary companies located at different countries (company A in S area and company B in T area), and the two companies want to share and interoperate their customers' information by Web Service. Let the description schemas are represented in XML format, and these schemas are deployed on their own XML web services. However, the XML schemas used by each company undergo periodic changes due to the dynamic nature of of its business. In order to the two companies be able to exchange and interoperate their information, semantic correspondences between elements of them should be identified. However, if has schema matching manually performed, it is a tiresome and costly process. Moreover, if the company A changes its customer information database structure, and the XML schema is changed synchronously, but they do not notice the company B to update its Web Service correspondingly, under this conditions, if the interoperate wants to carry out successfully, two web agents have to automatic matching their schemas again, and need not manual acting. The automatic schema matching can improve the reliability and usability of Web services. Now, two XML schemas are shown in Fig.2.1, which are based on BizTalk Schema specification, where, Schema S is used by company A, and Schema T is deployed by company B.*

Fig. 2.1 shows that the two schemas involve several heterogeneities that increase the difficulty of the schema matching problem. The complexity of schema matching arises due to the following reasons.

- *Representation problems*; This kind of problems arises from the fact that databases are engineered by different people. Even if two schemas are designed for the same domain, two problems can be recognized, (i) different possible representation models; different possible representation models can be chosen for the schemas and (ii) different possible names and structures; same concepts are represented using different names and structures, while the same name and structure could be used for different concepts. For example, the customer name in the first schema is "*AccountOwner*"while in schema T is "*Customer*". The name of the customer in schema S is modeled only by one element "*Name*" while in the second one is represented by two elements "*FName*" and "*LName*".

- *Semantic problems*; This kind of problems arises from the fact that the semantics of the involved elements can be inferred from only a few information sources, typically the data creators, documentation, and associated schema and data [51]. Extracting semantics information from these sources is often extremely bulky. Beside that, matching schema elements is based on clues in the schema and data. Examples of such clues include element names, types, data values, schema structures, and integrity constraints. However these clues are often unreliable and incomplete. For example, the element *Address* does not indicate whether it is a home or work address.

- *Computation cost problems*; To decide that an element s of schema S matches an element t of schema T, one must typically examine all other elements of T to make sure that there is no other element that matches s better than t. This global nature adds substantial cost to the matching process. To make matter worse, matching is subjective, depending on the application. Therefore the user must be involved in the matching process.

All these problems reveal an idea of the complexity inherent to the schema matching problem. In order to provide company users with consistent information, the schema matching system should identify that the elements *S.AccountOwner* and *T.Customer* are correspondent, as well as the element *S.AccountOwner.Name* is corresponding to two elements (*T.Customer.Fname, T.Customer.Lname*).

2.2 The schema matching problem

2.2.1 Problem definition

There are several definitions of matching and the schema matching problem [117, 30, 91, 63, 51, 127]. We here present a general definition, following the work in [11, 127]. We first introduce the definition of a generic match problem (GMP) then we cast this definition to the schema matching problem (SMP).

Definition 2.1 *The generic match problem (GMP) can be defined as 4-tuple element GMP=(S, T, P, O), where:*

- *S and T are the objects to be matched, where each object may have one or more elements,*
- *P is a set of predicate functions, having the values {true, false} and should be satisfied, and*
- *O is a set of objective functions, which determines the goodness of match.*

As mentioned, the goal of match function is to identify and discover semantically related elements between two objects S and T, such that all the predicate functions P are satisfied and values of object functions O are optimized. Since the solution of a matching problem is not unique but there exist a set of possible solutions depending on the application domain. Therefore, we should select the best of all possible solutions. An objective function, a function associated with an optimization problem, determines how good a solution is.

Example 2.2 *Let the objects S and T have the following elements {3, 7, AZ, 17} and {14}, respectively, and we want to identify corresponding elements between the two objects such that the elements of two objects have the same data type and the differences between S elements and T elements are very small. We can rewrite this generic matching problem as follows: GMP=(S, T, P, O) where:*

- *S={3,7,AZ,17}*
- *T={14}*
- *P : S and T elements have the same data type, i.e. P1: samedatatype(S.e, T.e)*

- *O: the difference between elements of two objects should be small as possible, i.e. O1: $|e.S - e.T|$ is minimum.*

A solution of the matching problem is a set of mapping elements which indicates that certain element(s) of S is (are) corresponding to certain element(s) of T satisfying all the conditions in the predicate sections with an optimized similarity value depending on the objective functions. For the above example, the predicate $P1$ shows that the elements 3, 7, and 17 of object S are corresponding with T element (have the same data type) but the objective function $O1$ explains why $S.17$ is the most appropriate corresponding for $T.14$ (we use here the dot notation). Since $|S.3 - T.14| > |S.7 - T.14| > |S.17 - T.14|$, i.e., $|S.17 - T.14|$ is the minimum.

Now we are in a position that the definition of the generic match problem (GMP) can be casted and used to define the schema matching problem (SMP).

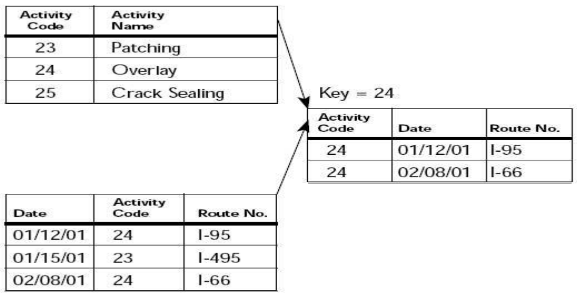
Definition 2.2 *The schema matching problem (SMP) is represented as 4-tuple element (S, T, P, O), where,*

- *S is the source schema*
- *T is the target schema*
- *P is a set of predicate functions, & O is a set of objective functions, as defined in the GMP.*

Formally, the matching process is represented as a function that accepts two schemas S and T as input in the presence of desired relations: predicate functions P and objective functions O and produces a set of mappings $\tilde{M}$. The mapping set, $\tilde{M}$, represents correspondences between S and T schema elements such that all predicates in P must be satisfied and the selected mappings are subjected to criteria in the objective functions O. Equation 2.1 expresses the matching process.

$$\tilde{M} = f(S, T, P, O) \tag{2.1}$$

In the following sections, we present schema matching environments: data models (input schemas) and matching output (mappings).



Activity Code	Activity Name
23	Patching
24	Overlay
25	Crack Sealing

Key = 24

Activity Code	Date	Route No.
24	01/12/01	I-95
24	02/08/01	I-66

Date	Activity Code	Route No.
01/12/01	24	I-95
01/15/01	23	I-495
02/08/01	24	I-66

Figure 2.2: Relational model [1]

2.2.2 Data model

A schema is the description of the structure and the content of a data model. It is a collection of meta-data that describes relations and a set of related elements in a database, such as tables, columns, classes, or XML elements and attributes. There are several kinds of data models, such as relational model, object-oriented model, XML schema, ontology, etc. By schema structure and schema content, we mean its schema-based properties and its instance-based properties, respectively. In the following, we present a number of various forms of data models.

Relational model. In the relational model, data items is organized as a set of formally described tables (relations) from which data can be accessed or reassembled in many ways without having to reorganize the database tables. Each table contains one or more data categories in columns. Each row contains a unique instance of data for the categories defined by the columns. A relational schema specifies the names of the tables as well as their types: the names and types of the columns of each table. The relational model also includes the notion of a key for each table: a subset of the columns that uniquely identifies each row. Finally, a column in a table may be specified as a foreign key pointing to a column in another table. This is used to keep referential constraints among various entities, as shown in Fig. 2.2.

Object-oriented data model. The object-oriented data model is based on the basic concepts of object-oriented, such as objects and objects' identifier, classes, encapsulation, etc. and benefits from the development in object-oriented analysis and design, object-oriented programming, and object-oriented distributed computing. Basic requirements of an object-oriented data model were listed in The Object-oriented Database System Manifesto [20]. In general, the data model defines a data object as containing code (sequences of computer instructions) and data (information that the instructions operate on). Traditionally, code and data have been kept apart. In an object-oriented data model, the code and data are merged into a single indivisible thing, an object, as shown in Fig. 2.3.

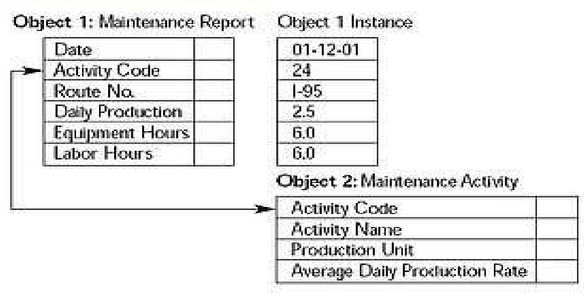
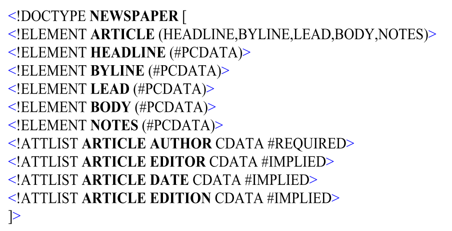


Figure 2.3: Object-oriented model [1]

XML. The eXtensible Markup Language (XML) is emerging as the de facto standard for information representation and exchange on the Web and on the Intranet. This is due to XML's inherent data self-describing capability and flexibility of organizing data [2]. First, XML data are self-describing, and are based on nested tags. Tags in XML data describe the semantic of data. This self-describing capability of XML data helps applications on the Web understand the content of XML documents published by other applications [71]. Moreover, the hierarchy formed by nested tags structures the content of XML data. The role of nested tags in XML is somewhat similar to that of schemas in relational databases. At the same time, the nested XML model is far more flexible than the relational model.

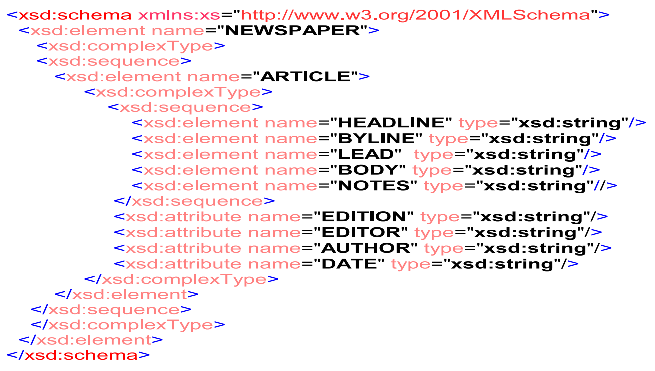
XML data can be broadly classified into two categories: XML schemas and XML documents. An XML document (document instance) represents a snapshot what the XML document contains. An XML schema is the description of the structure and the legal building blocks for an XML document. Several XML schema languages have been proposed [88]. Among them, XML document type definition (DTD) and XML Schema Definition (XSD) are commonly used.

XML DTD (DTD in short) is the de facto standard XML schema language of the past and present and is most likely to thrive until the arrival of XML Schema. Its main building block consists of an *element* and an *attribute*. It has limited capabilities compared to other schema languages. The real world is typically represented by the use of hierarchical element structures. XML Schema is an ongoing effort of



```
<!DOCTYPE NEWSPAPER [
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>
<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>
]>
```

(a) DTD



```
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="NEWSPAPER">
   <xsd:complexType>
   <xsd:sequence>
     <xsd:element name="ARTICLE">
        <xsd:complexType>
           <xsd:sequence>
              <xsd:element name="HEADLINE" type="xsd:string"/>
              <xsd:element name="BYLINE" type="xsd:string"/>
              <xsd:element name="LEAD"  type="xsd:string"/>
              <xsd:element name="BODY" type="xsd:string"/>
              <xsd:element name="NOTES" type="xsd:string"//>
            </xsd:sequence>
            <xsd:attribute name="EDITION" type="xsd:string"/>
            <xsd:attribute name="EDITOR" type="xsd:string"/>
            <xsd:attribute name="AUTHOR" type="xsd:string"/>
            <xsd:attribute name="DATE" type="xsd:string"/>
        </xsd:complexType>
     </xsd:element>
  </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

(b) XSD

Figure 2.4: Example of an DTD and its respective XSD.

W3C to aid and eventually replace DTD in the XML world. XML Schema aims to be more expressive than DTD and more usable by a wider variety of applications. It has many novel mechanisms such as simple and complex types, rich data type sets, occurrence constraints, and inheritance. Fig. 2.4 shows a simple DTD and its corresponding XML schema (XSD). The figure also illustrates the main differences between components of the two languages.

Ontology. The term ontology has its origin in philosophy, and has been applied in many different ways. The core meaning within computer science is a model for describing the world that consists of a set of types, properties, and relationship types [135]. Ontologies are used in artificial intelligence, the Semantic Web, software engineering, biomedical informatics, library science, and information architecture as a form of knowledge representation about the world or some part of it.

Contemporary ontologies share many structural similarities, regardless of the language in which they are expressed. The common components of ontologies include [63]:

- *Classes:* sets, collections, concepts, types of objects, or kinds of things are the main entities of an ontology;
- *Objects* or instances are interpreted as particular individual of a domain;
- *Attributes:* aspects, properties, features, characteristics, or parameters that classes can have;
- *Attribute instances:* are the properties applied to precise objects;

- *Relations:* ways in which classes and individuals can be related to one another;
- *Restrictions:* formally stated descriptions of what must be true in order for some assertion to be accepted as input;
- *Rules:* statements in the form of an if-then (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form ;
- *Axioms:* assertions (including rules) in a logical form that together comprise the overall theory that the ontology describes in its domain of application;
- *Events:* the changing of attributes or relations.

Ontologies can be expressed in an ontology language. There are several languages for representing ontologies [130], which can be broadly classified into: *traditional languages*, such as Frame Logic (F-Logic), *markup languages*, which use markup scheme to encode knowledge such as Ontology Web Language (OWL), or *structural languages* such as description logic-based.

Spotlight. The four mentioned data models, relational, object-oriented, XML, and ontology are currently surviving on the Web and on the Intranet and will remain so in the near future. However, in this thesis, we only consider schemas represented in the XML language. Our motivations for this selection are twofolds. The first is concerning XML features, while the second is related to XML-based applications. In terms of XML features, the self-describing capability and the flexibility of organizing data pave the way for the XML model to be a widely accepted data model in both scientific and commercial communities. As a result, the number of XML-based applications is rapidly increasing. There has been a growing need for developing high-performance techniques to efficiently manage XML data repositories.

XML schema representation

As XML is emerging as the data format of the internet era, there is an substantial increase of the amount of data in XML format. To better describe such XML data structures and constraints, several XML schema languages have been proposed. An XML schema is the description of a type of XML document, typically expressed in terms of constraints on the structure and content of that type [2]. There are several languages developed to express XML schema. The most commonly used are DTD and

XML Schema Definition (XSD) [88].

Spotlight. Both document type definition (DTD) and XML schema definition (XSD) are commonly used languages to represent XML data. However, DTD has limited capabilities compared to other schema languages, such as XSD. Moreover, XML Schema definition (XSD) aims to be more expressive than DTD and more usable by a wider variety of applications. It has many novel mechanisms, such as inheritance for attributes and elements, user-defined data types, etc. Therefore, the research, in this thesis, is only concerned with XML Schema Definition (XSD). Through this thesis, unless clearly specified, we used the term "schema" to present XML schema (XSD).

An XML schema is a set of schema components. In general, there are several kinds of components in all, falling into three groups[1].

- *Primary components,* which may or must have names and contain the following:
  - simple type definitions;
  - complex type definitions;
  - element declarations;
  - attribute declarations.

  The element and attribute declarations must have names, while the type definitions may have names.

- *Secondary components,* which must have names, are as follows
  - attribute group definitions;
  - identity constraint definitions;
  - model group definitions;
  - notation declarations;
  - type alternatives;
  - assertions.

- *Helper components,* which provide small parts of other components, they are not independent of their context and are as follows:

[1]http://www.w3.org/TR/xmlschema11-1/#components

- annotations;
- model groups;
- particles;
- wildcards;
- attribute use.

These components may be combined together to build an XML schema. The coexisting schemas range between small-scale and large-scale schemas. A small-scale schema is the schema of limited size and scope, while a large-scale schema is large in scope and extent.

Spotlight. The research in this thesis deals with both types of schemas.

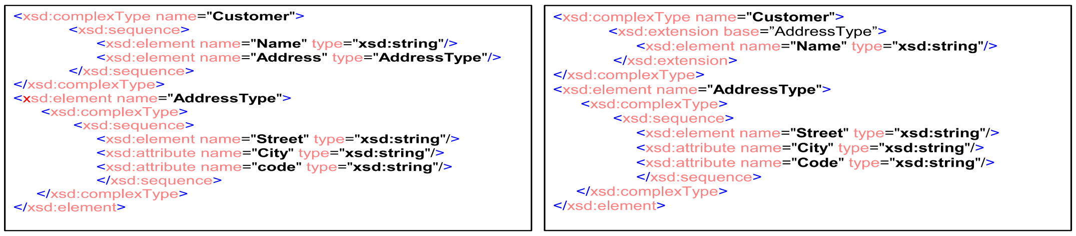
Issues in representing large-scale schemas

Advanced capabilities supported by the XSD language, such as user defined types, reuse of schema components, result in significant complications for schema matching [118, 48]. In this section, we discuss the issues related to representing large-scale schemas.

1. **XSD types**. The XSD language provides a flexible and adaptable types containing both simple and complex types. It contains several simple types (primitive and derived), which can be used for element and attribute declarations. Complex types are user-defined and can be used for element declarations. In contrast to simple types, complex types can have elements in their content and may carry attributes.

   There are two main methods to construct a new type either by a composition or by a derivation method. Fig. 2.5 illustrates the two methods supported by XSD. In the composition approach (Fig. 2.5a), a new type (*Customer*) is composed of elements/attributes of existing types (*string* and *AddressType*). Using derivation, a new type is derived from a base type and automatically inherits all its components. XSD supports two derivation mechanisms, namely extension and restriction. In Fig. 2.5b, type *Customer* extends type *AddressType* and inherits the elements *Street, City* and *Code* of *AddressType.*

   A schema matching system presents high-performance if it has the ability to exploit XSD types information. Thus, different algorithms should be developed to determine the similarity between XSD types across different schemas. Measuring



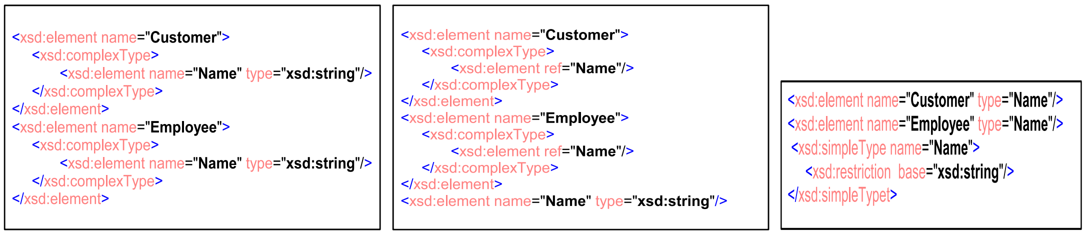
```
<xsd:complexType name="Customer">
        <xsd:sequence>
            <xsd:element name="Name" type="xsd:string"/>
            <xsd:element name="Address" type="AddressType"/>
        </xsd:sequence>
</xsd:complexType>
<xsd:element name="AddressType">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Street" type="xsd:string"/>
            <xsd:attribute name="City" type="xsd:string"/>
            <xsd:attribute name="code" type="xsd:string"/>
            </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

(a) Composition

```
<xsd:complexType name="Customer">
        <xsd:extension base="AddressType">
            <xsd:element name="Name" type="xsd:string"/>
        </xsd:extension>
</xsd:complexType>
<xsd:element name="AddressType">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Street" type="xsd:string"/>
            <xsd:attribute name="City" type="xsd:string"/>
            <xsd:attribute name="Code" type="xsd:string"/>
            </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

(b) Derivation by extension

Figure 2.5: XSD new types construction.



```
<xsd:element name="Customer">
   <xsd:complexType>
       <xsd:element name="Name" type="xsd:string"/>
   </xsd:complexType>
</xsd:element>
<xsd:element name="Employee">
   <xsd:complexType>
       <xsd:element name="Name" type="xsd:string"/>
   </xsd:complexType>
</xsd:element>
```

(a) Inline

```
<xsd:element name="Customer">
   <xsd:complexType>
       <xsd:element ref="Name"/>
   </xsd:complexType>
</xsd:element>
<xsd:element name="Employee">
   <xsd:complexType>
       <xsd:element ref="Name"/>
   </xsd:complexType>
</xsd:element>
<xsd:element name="Name" type="xsd:string"/>
```

(b) Element reuse

```
<xsd:element name="Customer" type="Name"/>
<xsd:element name="Employee" type="Name"/>
 <xsd:simpleType name="Name">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
```

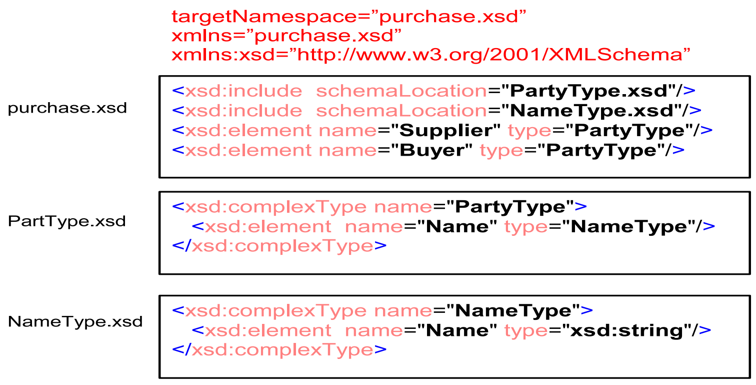
(c) Type reuse

Figure 2.6: XSD components reuse.

the similarity between simple data types is a simple task by providing a data type compatibility table [96]. However, determining the similarity among complex types increases the difficulty of schema matching [118, 48].

2. **XSD components reuse**. To avoid redundancy, especially in large-scale schemas, XSD supports two types of schema components reuse, as shown in Fig.2.6. The first is *element reuse*, wherein XSD components, such as elements, attributes, types can be referenced in several places. The referenced components are only limited to global components, which are immediate children of the schema root element. The second is *type reuse*. Types can be referenced within element or attribute declarations as well as (recursively) within other type definitions. The high flexibility of type reuse makes it a well-suited approach for large business applications. Both kinds of schema components reuse, shown in Fig. 2.6(b,c), result in representing XML schemas in graph structures. However, the basic method of element and attribute declarations is to to specify types inline, as shown in Fig. 2.6a, resulting in a tree-like structure schemas.

   Again, a high-performance matching system should be able to deal with different design methods (inline, element reuse, and type reuse). This requires the matching



```
targetNamespace="purchase.xsd"
xmlns="purchase.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
<xsd:include  schemaLocation="PartyType.xsd"/>
<xsd:include  schemaLocation="NameType.xsd"/>
<xsd:element name="Supplier" type="PartyType"/>
<xsd:element name="Buyer" type="PartyType"/>
```

```
<xsd:complexType name="PartyType">
  <xsd:element  name="Name" type="NameType"/>
</xsd:complexType>
```

```
<xsd:complexType name="NameType">
  <xsd:element  name="Name" type="xsd:string"/>
</xsd:complexType>
```

Figure 2.7: Distribution of a schema [118, 48]

system uses a uniform schema representation, which is not biased to any design style and can cope with shared components. Shared components are important for schema matching, but also are difficult to deal with. On the other hand, it may be important to clearly differentiate between the contexts where a shared component is used, e.g. to distinguish the names of customers vs. employees.

3. ***XSD schema plans.*** In small-scale XML-based applications, the plan to construct a schema is to put all schema components in one schema document. However, the XSD language supports the distribution of a schema over schema documents and namespaces through the *include* and *import* directives. The import directive allows to add multiple schemas with different target namespaces to a schema document, while the include directive provides adding multiple schemas with the same target namespaces, as shown in Fig. 2.7. The figure shows that all documents declare the same target namespace purchase.xsd. The main document purchase.xsd, references type PartyType defined in document PartyType.xsd, which in turn references type NameType in document NameType.xsd [118].

   Consequently, a high-performance matching matching system should be able to deal with the distribution of a schema over schema documents.

Spotlight. The illustrated issues in representing large-scale schemas as well as heterogeneities discussed in the previous section amplify challenges towards developing high-performance schema matching techniques. Our interest, in this thesis, is to deal with these challenges. In particular, we aim to develop a generic schema matching framework that is able to cope with several performance aspects including matching effectiveness (quality) and matching efficiency (scalability) and has the advantage of dealing with complex matching.

2.2.3 Matching output

As mentioned before, the main objective of XML schema matching is to identify and discover semantic correspondences among elements across XML schemas. The semantic correspondences between these elements can be represented as a set of mapping elements. A mapping element should comprise the corresponding elements and the relation that is supposed to hold between them. Given two schemas S and T, a mapping element can be specified by a 4-tuple element [8]

$$< ID, S.\mathcal{E}\ell_i, T.\mathcal{E}\ell_j, R > \tag{2.2}$$

where

- ID is an identifier for the mapping element;
- $S.\mathcal{E}\ell_i \in S$ and $T.\mathcal{E}\ell_j \in T$ are the corresponding elements. It is worth noting that the element, $\mathcal{E}\ell$, means either simple or complex type definition or element or attribute declaration. A formal definition of the element will be detailed in Chapter 5.
- R indicates the similarity value between 0 and 1. The value of 0 means strong dissimilarity while the value of 1 means strong similarity.

For example, after applying a certain matching algorithm to XML schemas shown in Fig. 2.1, the similarity value between the two elements “S.Address” and “T.CAddress” could be 0.8. Suppose that this matching algorithm uses a threshold of 0.3 for determining the resulting match, i.e., the algorithm considers all the pairs of elements with a similarity values higher than 0.3 as correct correspondences. Thus, we write this mapping element as follows:

$$\tilde{m_1} =< id_1, S.Address, T.CAddress, 0.8 > \tag{2.3}$$

A mapping element may have an associated mapping expression, which specifies how the two elements (or more) are related. Schema matching is only considered with identifying the mappings not determining the associated expressions. Discovering the associated mapping expressions is another related problem called mapping discovery [103].

We distinguish between two types of matching, which are defined as follows. Let S and T be two schemas having n and m elements respectively, the two types are:

- *Simple matching*; For each element $\in S$, find the most semantically similar element $\in T$. This problem is referred to as *one-to-one matching*. For example, the correspondence between the "S.street" and "T.street" elements in the two XML schema shown in Fig.2.1 represents a simple matching.
- *Complex matching*; For each element (or a set of elements) $\in S$, find the most semantically similar set of elements $\in T$. The correspondence between the "S.Name" element of the first schema shown in Fig.2.1 and the "T.(FName, LName)" elements of the second schema shown in the figure represents a complex matching.

2.3 Summary

In this chapter, we discussed different aspects of schema matching. To motivate the schema matching problem we first introduced a real example from the e-business domain, which showed the importance of schema matching. Then, we defined the generic matching problem and casted this definition to define the schema matching problem. From this definition, we illustrated the matching process input and output. We focused on different data models that represent input schemas, and gave more attentions to XML schemas identifying what different heterogeneities are and what issues in representing large-scale schemas are and why they increase the complexity of schema matching.

Part II

Schema matching: The state of the art

3 Application domains

As mentioned in Chapter 2, the activity that discovers semantic correspondences between elements across different XML schemas is called schema matching. Schema matching plays a crucial role in many data-shared applications that need to exchange data between them. These applications make use of schema matching techniques during either design time or run time. Typically, schema matching plays the central role: in *data integration* to identify and characterize inter-schema relationships across multiple (heterogeneous) schemas; in *data warehousing* to map data sources to a warehouse schema; in *E-business* to help to map messages between different XML formats; in the *Semantic Web* to establish semantic correspondences between concepts of different ontologies; in *data migration* to migrate legacy data from multiple sources into a new one; and in *XML data clustering* to determine semantic similarities between XML data.

In this chapter, we first present some well-known applications where matching has been recognized as a plausible solution for a long time. These are data integration, data warehouse, and schema evolution. We then discuss some recently emerged applications, such as peer-to-peer information sharing, web service discovery, XML data clustering, and query answering on the Web.

3.1 Data integration

XML is widely used for the exchange of data among Web applications and enterprises. Integration of distributed XML data is thus becoming a research problem. This is due to the large number of business data appearing on the Web and the large number of service-oriented architecture that is being adapted in the form of Web services. XML data integration includes the construction of a global view for a set of independently developed XML data [25, 87, 32, 136].

Since XML data are engineered by different people, they often have different structural and terminological heterogeneities. The integration of heterogeneous data sources requires many tools for organizing and making their structure and content homogeneous. XML data integration is thus a complex activity that involves reconciliation at different levels:

1. *Data models*. Integrating XML data with other data sources can greatly differ with respect to the structures they use to represent data (for example, tables, objects, files, and so on) [83]. Reconciliation of heterogeneous data models requires a common data model to map information coming from the various data sources.

2. *Data schemas*. Once we have agreed upon a common data model, the problem arises of reconciling different representations of the same entity or property. For example, two sources may use different names to represent the same concept (price and cost), or the same name to represent different concepts (project to denote both the project an employee is working on and a project for which an employee is the reviewer), or two different ways for conveying the same information (date of birth and age). Additionally, data sources may represent the same information using different data structures. For instance, consider two data sources that represent data according to the relational model, where both sources model the entity *Employee* but the first uses only one table to store employee information while the other spreads this information across more than one table. The need thus arises for tools to reconcile all these differences [89].

3. *Data instances*. At the instance level, integration problems include determining if different objects coming from different sources represent the same real-world entity and selecting a source when contradictory information is found in different data sources (for instance, different birth dates for the same person) [52].

Moreover, the integration of the Web data increases the integration process challenges in terms of heterogeneity of data. Such data come from different resources and it is quite hard to identify the relationship with the business subjects.

Usually, a first technical step is to identify correspondences between semantically related elements of the schemas. Then, by using the identified correspondences, merging the databases is performed. The matching step is still required, even if the databases to be

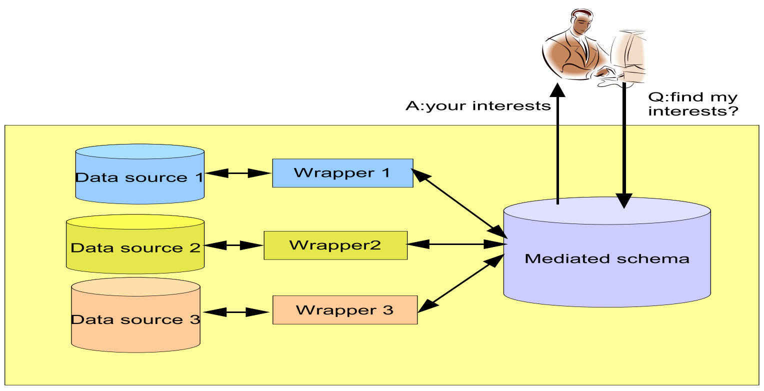


Figure 3.1: Simple schematic for a data integration system.

integrated are coming from the same domain of interest, e.g., book selling or car rentals. This is because the schemas have been designed and developed independently. In fact, humans follow diverse modeling principles and patterns, even if they have to encode the same real-world object. Finally, the schemas to be integrated might have been developed according to different business goals. This makes the matching problem even harder.

There are several directions of using matching in data integration. The recent trend in data integration has been to loosen the coupling between data. Here, the idea is to provide a uniform query interface over a mediated schema (see Fig. 3.1). This query is then transformed into specialized queries over the original databases. To answer queries, the data integration system uses a set of semantic mappings between the mediated schema and local schemas of data sources. The system uses the mapping to reformulate a user query into a set of queries on the data sources. Wrapper programs, attached to each data source, handle the data formatting transformation between local schemas and the mediated schema [50, 114]. This process can also be called view-based query answering because we can consider each of the data sources to be a view over the (non-existent) mediated schema [74]. Formally, such an approach is called Local As View (LAV), where "Local" refers to the local sources/databases. An alternate model of integration is one where the mediated schema is designed to be a view over the sources. This approach is called Global As View (GAV), where "Global" refers to the global (mediated) schema, is often used due to the simplicity involved in answering queries issued over the mediated schema.

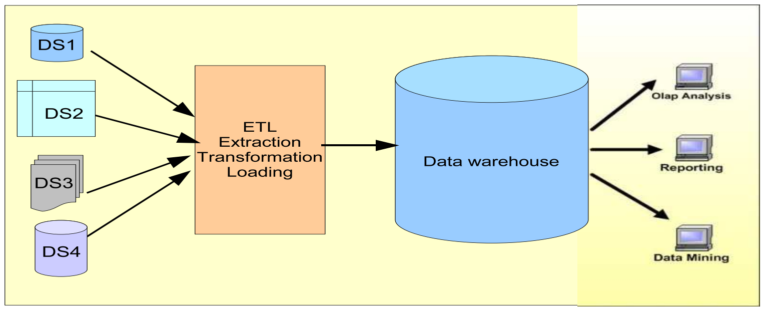


Figure 3.2: Data warehouse architecture.

3.2 Data warehousing

A data warehouse is a centralized repository of an organization's electronically stored data from multiple information sources that are transformed into a common, multidimensional data model for efficient querying and analysis. This definition of the data warehouse focuses on data storage. However, the means to retrieve and analyze data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system, see Fig. 3.2. The extraction process requires transforming data from the source format into the warehouse format. As shown in [31, 24], schema match is a useful process for designing transformations. Given a data source, one approach to create appropriate transformations is to start by finding those elements of the source that are also present in the warehouse. This is a match operation. After an initial mapping is created, the data warehouse designer needs to examine the detailed semantics of each source element and create transformations that reconcile those semantics with those of the target. Another approach to integrate a new data source is to reuse an existing source-to-warehouse transformation. First, the common elements of the new data source and the old one are found (a match operation) and then, the existing source-to-warehouse transformation is reused for those common elements.

The increasing use of XML in business-to-business (B2B) applications and e-Commerce Web sites, suggests that a lot of valuable external data sources will be available in XML format on the Internet. External XML data include business documents like purchase orders, lists of prices or catalogs, as well as Web service responses. The possibility of integrating available XML data into data warehouses plays an important role in providing enterprise managers with up-to-date and comprehensive information about their business domain [137, 116].

3.3 Schema evolution & data migration

Schema evolution is the process of assisting and maintaining the changes to the schematic information and contents of a database. An important issue in schema evolution is to provide evolution transparency to the users, whereby they would be able to pose queries to the database based on a (possibly old) version of the schema they are familiar with, even if the schema has evolved to a different state.

In the XML context, XML data tend to change over time for a multitude of reasons, including the correction of design errors of XML schemas, allowing the expansion of the application scope over time, or the accounting for merging of several businesses into one [86, 72, 104]. These situations arise because knowledge engineers and developers usually do not have a global view of how and where the schemas have changed. Therefore, developers need to manage and maintain the different versions of their schemas. The goal of the matching process is to discover correspondences between the old schema and the new version of the schema. These correspondences are then used to generate transformation rules for the underlying XML document instances.

Another application related to schema evolution is data migration. A common task in many database applications is the migration of legacy data from multiple sources into a new one. This requires identifying semantically related elements of the source and target systems (schema matching) and the creation of mapping expressions (mapping discovery) to transform instances of those elements from the source format to the target format [58].

3.4 Peer-to-peer systems

Data management in distributed systems has been traditionally achieved by distributed database systems [111] which enable users to transparently access and update several databases in a network using a high-level query language. A distributed database system is a centralized server that supports a global schema and implements distributed database techniques (query processing, transaction management, consistency management, etc.). This approach has proved effective for applications that can benefit from centralized control and full-fledge database capabilities, e.g., information systems. However, it cannot scale up to more than tens of databases [136]. In contrast, peer-to-peer (P2P) systems, a fully distributed communication model in which parties (called peers) have equivalent functional capabilities in providing each other with data and services [146], suggest a scalable, decentralized and easily extensible integration architecture where any peer can

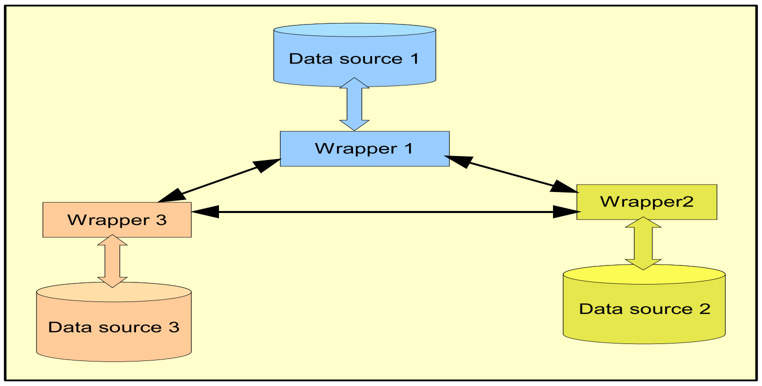


Figure 3.3: Peer-to-peer system.

contribute data, schemas, and mappings. Peers with new schemas simply need to provide a mapping between their schema and any other schema already used in the system to be part of the network, see Fig. 3.3.

Generally, peer-to-peer management systems generally use peer-to-peer overlay networks to support their distributed operations [141]. Some use unstructured overlay networks [75] to organize the peers into a random graph and use flooding or random walks to contact distant peers. Other peer systems maintain a decentralized yet structured peer-to-peer network [42] to allow any peer to contact any other peer by taking advantage of a distributed index.

To support query answering by peer-to-peer systems, there are key-based, keyword-based, and schema-based systems [131]. Key-based systems can retrieve information objects based on a unique hash key. This kind of queries is supported by all Distributed Hash Table (DHT) networks. Keyword-based systems extend this to the possibility to look for the document based on a list of query terms. Schema-based systems manage and provide query capabilities for structured information, such as relation and XML databases. A schema matching process should be performed to automatically discover mappings between pairs of schemas to iteratively disseminate a query from one database to all the other related databases [43].

3.5 XML data clustering

The increasing availability of heterogeneous XML data has raised a number of issues concerning how to represent and manage semi-structured data. As a result, discovering knowledge to infer semantic organization of XML data has become a major challenge in

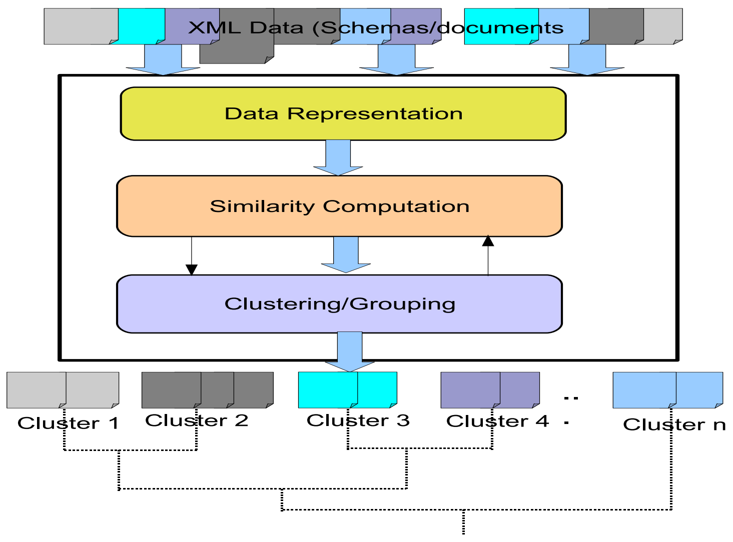


Figure 3.4: A generic XML data clustering framework.

XML data management. A possible solution is to group similar XML data based on their content and structures. Grouping similar XML data according to structure or content or both among heterogenous set of data is the process of *XML data clustering*.

Clustering is a useful technique for grouping data objects such that objects within a single group/cluster have similar features, while objects in different groups are dissimilar [82]. The main steps involved in the data clustering activity, as shown in Fig. 3.4, are: (1) *data representation*: data objects are represented using a common data model; (2) *definition of data proximity measures suitable to the data domain and data representation*: data features and the proximity function to measure the similarity between pairs of data objects are determined; and (3) *clustering or grouping*: the similar data objects are grouped together based on the proximity function using clustering algorithms [82, 28].

The relationship between schema matching and XML data clustering is bidirectional. On the one side, clustering techniques have been adopted to improve matching performance [93, 127, 122], and on the other side, schema matching is the backbone of the clustering technique [89, 108, 13]. The SemInt system [93] starts by building the attribute signatures, and then uses an unsupervised learning algorithm to classify the attributes within the database. The Self-Organized Map algorithm is used to serve as clustering algorithm to classify attributes into different categories. The output of this classifier is used as training data for a neural net. The back-propagation learning algorithm is used to train a network to recognize input patterns and give degrees of similarities. The work proposed in [127] is based on using a clustering technique. The technique is inspired by an observation that for a certain personal schema, good mappings are often found in limited areas within the schema repository. Clustering is used as a tool to quickly identify such areas and then restrict the expensive schema matching algorithms to these areas instead of searching through the whole repository. This technique, called clustered schema matching, adds a clustering step to an existing schema matching system. PORSCHE [122] first clusters XML schema elements based on a linguistic label similarity making use of a domain-specific user-defined abbreviation table and manually defined domain-specific synonym table.

On the other hand, schema matching can be used as a technique that measures the similarity between XML data. Approaches proposed in [89, 108, 13] consider XML data element details such as name, data type as well as the relationships between elements. The approaches compute the similarity between XML data based on the computed similarities among schema elements. These element-level similarities are then aggregated in order to determine the semantic similarities among paths and XML data trees.

3.6 Web service discovery

The Web, once solely a repository for text and images, is evolving into a *provider of services*, such as flight information providers, temperature sensors, and *world-altering services*, such as flight booking programs, and a variety of e-commerce and business-to-business applications [99]. These web-based applications rely heavily on permitting generic software components to be developed and shared. With the adoption of XML as a standard and a common language for information representation and exchange on the Web, the underlying principles have gained wide scale adoption through the definition of Web service standards. Web services are well-defined, reusable, software components that perform specific, encapsulated tasks via standardized Web-oriented mechanisms [38]. The common usage scenario for Web services, as shown in Fig. 3.5, can be defined by three phases; *Publish, Find* and *Bind*; and three entities: the service requester, which invokes services; the service provider, which responds to requests; and the registry, where services can be published.

Three fundamental layers are required to provide or use Web services [22]. First, Web services must be network-accessible to be invoked, HTTP is the de-facto standard network protocol for Internet-available Web services. Second, Web services should be XML-based

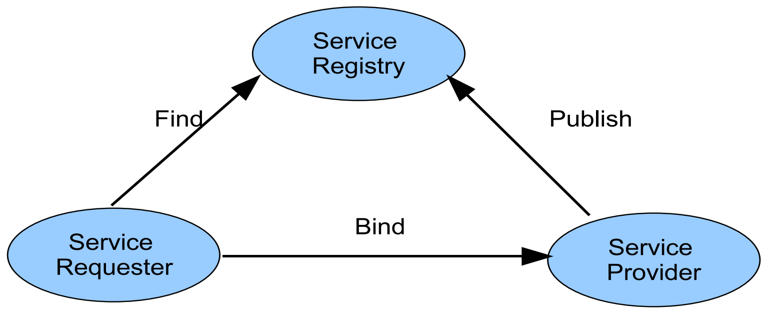


Figure 3.5: Web service usage scenario [38]

messaging for exchanging information, and SOAP[1] is the chosen protocol. Finally, it is through a service description that all the specifications for invoking a Web service are made available; WSDL[2] is the de-facto standard for XML-based service description.

Web service discovery mechanisms allow access to service repositories. These mechanisms should offer a number of capabilities, recognizable at both development and execution time. During development, one may search repositories for information about available services. At execution, client applications may use repositories to discover all instances of a Web service that match a given request. Normally, Web services in a given repository are developed independently, i.e., they are heterogeneous. This heterogeneity is considered the main obstacle affecting the discovery process. A match task is required to identify and discover the best Web service available. Different approaches have been developed to discover Web services based on schema matching techniques [76, 95, 16].

3.7 XML Query processing

Although XML is usually used as an information exchange standard, storing, indexing and querying XML data are still important issues and have become research hotspots both in the academic community and in the industrial community [139, 71]. So far, there have been three main approaches to manage XML data: (1) *the relational model*, mapping XML data into tables and translating queries on XML data into SQL statements [64, 123], (2) *the object-oriented model*, storing XML documents into an OODB with classes and translating XML queries into OQL queries based on XML data schema information [119], and (3) *the native model*, designing special structures and indexes to store and index XML data and proposing particular query optimization techniques to query XML data [100].

[1]http://www.w3.org/TR/soap/
[2]http://www.w3.org/TR/wsdl20/

XML queries can be categorized into two classes: *database-style* queries and *Information Retrieval-style queries*. Database-style queries return all query results that precisely match (the content and structure requirements) the queries, while, IR-style queries allow "fuzzy" query results, which are ranked based on their relevance to the queries [71]. Queries using XML query languages make use of twig patterns to match relevant portions of data in an XML data source. Twig patterns are simple tree-structured queries for XML that include three basic language elements, namely node conditions, parent-child edges, and ancestor-descendant edges. They play a very important role in database-style languages, such as XPath and XQuery as well as information retrieval-style approaches to match relevant portions of data in an XML database [143, 71].

Given a twig pattern T and an XML database D, a match of T in D is identified by a mapping from nodes in T to elements in D, such that: (i) query node predicates are satisfied by the corresponding database elements; and (ii) the parent-child and ancestor-descendant relationships between query nodes are satisfied by the corresponding database elements. The answer to query T with m nodes can be represented as a list of m-ary tuples, where each tuple $(t_1, ..., t_m)$ consists of the database elements that identify a distinct match of T in D.

3.8 Summary

To motivate the importance of schema matching, in this chapter, we reported on several shared-data XML-based applications from different domains. The common feature of these applications is that their XML data are developed independently, which results in semantic and structure heterogeneities. It becomes inevitable to consider schema matching to resolve these heterogeneities and provide a unified view of data. However, there are notable differences in the way these applications use matching. One interesting point is the static or dynamic aspect of data interoperability. Query answering and peer-to-peer systems have a more dynamic nature compared to the other application domains. The fact that peers in peer-to-peer systems have the ability to enter or leave the system at any time increases the dynamism of matching.

4
Schema matching systems: An overview

As shown in Chapter 3, schema matching represents the backbone of several XML-based applications. It has been shown that these applications make use of schema matching either during the design time or during the run time. The emergence of numerous matching systems motivates the need to make a survey on these systems [117, 63, 110, 51, 125]. To this context, this chapter is devoted to present an overview of existing matching systems. The main aim of the chapter is not to present a comparison between matching systems, but to introduce a unified view of these systems. To this end, we propose a generic matching framework. The framework should be independent from the representation scheme of the schemas to be matched as well as independent on the domain of these models. The aim is to draw a clear picture on the approaches, how far we can go with the current solutions, and what remains to be achieved. Here, the focus is on the concepts, alternatives, and fundamentals of the solutions, not on detailed technical discussions.

4.1 Generic schema matching framework

The existing matching systems deal with the schema matching problem from different point of views, depending on the schemas to be matched, which kind of schemas to be matched (relational, XML, ontology); used algorithms, the used methods to determine semantic correspondences between schema elements (rule-based or learner-based); application domains, etc. There has been a growing need for developing generic frameworks that unify the solution to the schema matching problem. The intuition behind introducing a generic framework is that recently existing matching systems and prototypes addressing

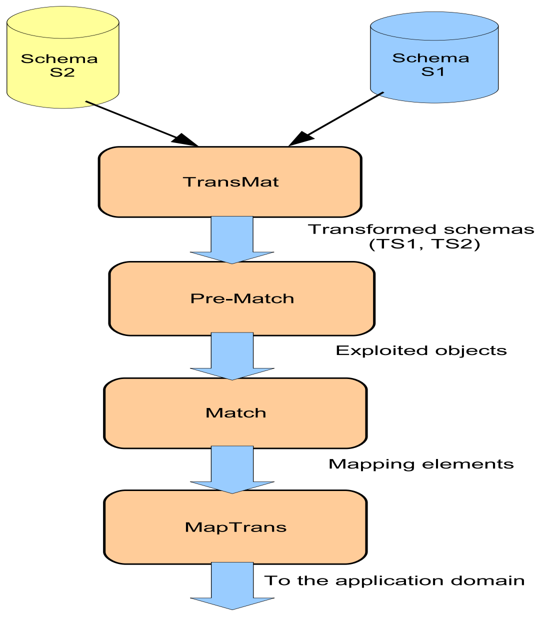


Figure 4.1: Schema matching phases

the schema matching problem have common similarities in the way of dealing with the problem. We observed that most of these approaches share the same phases but each phase may have its own steps depending on the nature of the approach. Therefore, in this chapter, we propose a unified framework, as shown in Fig. 4.1. The framework consists of the following general phases:

1. *TransMat*; Schemas to be matched should be imported and transformed into a common data model,
2. *Pre-Match*; Elements of common models exploited by matching algorithms should be identified,
3. *Match*; Matching algorithms are then applied to identified elements,
4. *MapTrans*; The match result is exported to the specified application domain.

In the following sections, we strive to give more details about these phases. Throughout the framework phases, we quote from existing mainstream systems. In fact, the number of existing matching systems is extremely large and it is difficult to mention all these systems in the thesis. Some reviews of a number of matching systems have already be presented, in particular in [117, 46, 63, 125, 110, 51]. Therefore, we only consider well-known matching systems that emerged during the last decade. Furthermore, since the main focus of the thesis is on schema-based matching, a few instance-based and ontology matching systems will be discussed. In particular, we consider Cupid [96], COMA/COMA++ [47, 118, 21, 48], Similarity Flooding (SF)/Rondo [101, 102], SemInt [93], LSD/GLUE [50, 49, 54, 53], Quick ontology mapping (QOM) [62], Ontobuilder [68, 66], Spicy [34, 35], S-Match [124, 70], Bellflower [128, 127], BTreeMatch [61, 60], PORSCHE [122], iMAP [44], and Dual Correlation Mining (DCM) [77]. Though different in many ways, these systems can be described through our generic framework.

4.2 Transformation for matching (TransMat)

To make the matching process a generic process, schemas to be matched should be represented internally by a common representation. This uniform representation reduces the complexity of the matching process by not having to cope with different schema representations. By developing such import tools, the schema match implementation can be applied to schemas of any data model such as relational, object-oriented, XML, UML, etc. Therefore, a first step in almost all schema matching systems is to transform input schemas into a common model in order to apply the matching algorithm. We name this phase *TransMat*; Transformation for Matching process.

Most current schema matching systems choose graph data structure as the internal representation [48, 101, 37, 128, 147]. The choice of graph as an internal representation for the schemas to be matched has many motivations. First, graphs are well-known data structures and have their algorithms and implementations. Second, by using the graph as a common data model, the schema matching problem is transformed into another standard problem, graph matching. Though they achieve high matching quality, matching systems depending on graph representation have poor matching performance, especially when they are used in a large-scale context. Therefore, other matching systems model input schemas as trees [89, 122, 60]. In fact, a schema is a graph. It can be represented as a tree by dealing with nesting and repetition problems using a set of predefined transformation rules similar to those in [89]. In the sequel, we focus on how the mainstream systems internally represent their input schemas.

Cupid [96] is a generic schema matching prototype that identifies mappings between XML schemas. It first restricts the model with hierarchical schemas, so it transforms its model into a schema tree. To deal with real-world schemas, Cupid extends its techniques to transform schemas into rooted graphs whose nodes represent schema elements. COMA/COMA++ [47, 48] are two systems for combining match algorithms in a flexible way. Both deal with different kinds of schema models, such as XML, RDF, and OWL schemas. Input schemas are represented by rooted directed acyclic graphs, where schema elements are represented by graph nodes connected by directed links of different types. The SF system [101] is a simple structural algorithm, based on a fixpoint computation, and can be used for matching of diverse data models. The system transforms the original schemas into directed labeled graphs using an import filter. The representation of the graphs is based on the Open Information Model (OIM) specification [29]. The system uses two different types of nodes. Oval nodes denote the identifers of nodes, and rectangle nodes denote string values. S-Match [70], Ontobuiler [68], and QOM [62] are well-known prototypes proposed to identify semantic correspondences between ontologies. They use graphs as the internal representation of input ontologies.

Like our proposed approach, BTreeMatch [61], PORSCHE [122], and Bellflower [128, 127] are three proposed prototypes identifying semantic correspondences across XML schemas. Both, BTreeMatch and PORSCHE encode input XML schemas as labeled trees, while the Bellflower system uses directed graphs enriched with node and edge labels. The Spicy system [34, 35] deals with both flat (i.e., relational) sources, and nested ones, as XML repositories. To represent input sources as a uniform representation, it utilizes the tree structure. The SemInt system [93] extracts metadata directly from the databases and DBMSs without needing to transform the original schemas. The system is concerned with attribute names, attribute values and domains and field specification. Then it uses this information to form attribute signatures. The LSD [50, 53] system delays the extraction and transformation phase into the next phase, i.e., LSD performs the first two phases (TransMat and Pre-Match) at the same time.

DCM [77] is a framework to discover complex matchings across Web query interfaces by a correlation mining approach. It views each query interface as a flat schema with a set of attribute entities. An attribute can be identified by attribute name, type, and domain.

From this, it is clear that a first step to construct a generic schema matching system, is to represent input schemas internally by a common data model. This common model should capture several kinds of model information, including syntactic, semantic, and structural information. The common model also allows matching systems to deal with different models from different domains.

4.3 Pre-Match phase

The pre-match phase is an important step in the matching process. Its importance comes from its location directly before the matching phase. This means that its output affects the input of the matching phase. Therefore, it should be examined very carefully. In addition, the nature of this phase depends on the type of matching algorithms. The matching algorithms can be classified as *rule-based* or *learner-based* [51].

In the rule-based algorithms, matching systems depend on hard-coded fashion. The Pre-Match phase does not appear clearly but can indirectly come into sight in the next phase. COMA/COMA++ renames this phase as *element identification*. The graphs are traversed to identify schema graph elements that are exploited by match algorithms to determine the similarity between them. Besides nodes as a common element type, COMA/COMA++ also supports matching between paths and fragments. To identify elements exploited by matching algorithms, Cupid also traverses schema graphs in a combined bottom-up and top-down manner. In the SF system, the Pre-Match phase does not appear directly. The system obtains an initial mapping called "$initialMap$" between the graphs using an operator called StringMatch. $initialMap$ is obtained using a simple string matcher that compares common prefixes and suffixes of string values. This initial mapping is used as a starting point for the next step. To compare two entities from two different ontologies, QOM considers their features through two steps: feature engineering and search selection. These features have to be determined by an expert understanding the encoded knowledge in ontologies.

The Bellflower prototype extends the graph representation of XML schemas and formalizes the schema matching problem as a constraint optimization problem [128]. The Spicy system also extends tree representation and generates an *electrical circuit* for every data tree. In order to build such circuit, it needs to sample instances, such that for a leaf node in the data tree, the system selects random instances with a defined size. The Spicy system then investigates features, such as the length of values in the sample and the distribution of characters.

In the learner-based algorithms, the matching system depends on Artificial Intelligence (AI) techniques; namely *machine learning* and *neural networks*. The Pre-Match phase is called the training phase and it is performed explicitly in most matching systems exploiting learner-based algorithms. The SemInt system uses a neural network learner. In the training (Pre-Match) phase, after building the attribute signatures, the system uses an unsupervised learning algorithm to classify the attributes within the database. The

Self-Organized Map [84] algorithm is used to serve as a clustering algorithm to classify attributes into different categories. Then, the output of this classifier is used as training data for a neural net. The back-propagation [120] learning algorithm, a supervised learning algorithm, is used to train a network to recognize input patterns and give degrees of similarities. The LSD/GLUE systems use a machine learning learner. The systems incorporate the transformation and the pre-match phases into one phase named the training phase. In LSD, the training phase consists of five steps starting by asking the user to specify 1-1 mappings which can be used as training data for the learner. Then, LSD/GLUE extracts data from the sources. After that, the LSD system uses this extracted data together with 1-1 mappings provided by the user to create the training data for each base learner. The system uses different base learners, e.g., name matcher and Naive Bayes learners and only one meta learner. Next, the LSD system starts training each base learner and then ends with training the meta-learner.

The iMAP system does not need a Pre-Match phase since it is restricted to relational tables. It generates a set of match candidates guided by a set of search modules. DCM performs a data preprocessing step to make input schemas ready for mining. The data preprocessing step consists of attribute normalization, type recognition to identify attribute types from domain values, and syntactic merging to measure the syntactic similarity of attribute names and domain values.

An interesting point that should be considered during the Pre-Match phase is the preparation of auxiliary information sources, such as external dictionaries and thesaurus, synonym tables, etc. Cupid makes use of a synonym and hypernym thesaurus to measure the name similarity, while COMA/COMA++ uses external dictionaries to measure the synonym between schema elements and user-specified matches to support user feedback. PORSCHE utilizes a domain-specific user-defined abbreviation table and a manually defined domain-specific synonym table to determine a linguistic label similarity between tree nodes. S-Match exploits WordNet to compute the meaning of a label at a node (in isolation) by taking a label as input, and by returning a concept of the label as output.

To summarize, the Pre-Match phase is an intricate and important phase. Its importance arises from the fact that it is located directly before the match phase and its output largely affects the match phase performance. Despite its importance, the work done toward this phase is very modest, especially the rule-based matching systems.

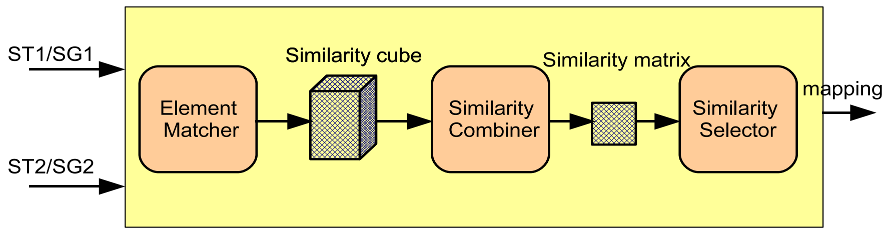


Figure 4.2: Match phase steps

4.4 Match phase

To identify semantic correspondences between schema elements, we need to assess their semantic similarities. To compute the semantic similarity across schema elements, matching systems use heuristics that are also called *clues*. Clues, in general, are often incomplete. Therefore, matching approaches use multiple clues to improve the probability that each matching situation at least some clues will correctly detect semantic similarity. Figure 4.2 shows the block diagram of the matching phase. It mainly consists of three steps:

1. *Element matcher;* It computes a similarity degree between every pair of schema elements using multiple heuristics. The output of this step is called the *similarity cube*.
2. *Similarity combiner;* It combines the different similarities for each pair of schema elements into a single similarity value using the similarity cube. The output of this step is called the *similarity matrix*.
3. *Similarity selector;* It selects the most plausible corresponding elements.

Below, we present more details about these steps according to the mainstream matching systems reported in this chapter.

4.4.1 Element matcher

The implementation of matching system heuristics is the element matcher. To distinguish between matchers, as shown in Fig. 4.3, two aspects are discussed:

- *Element properties;* The element properties are the properties used by the matcher to compute the element similarity. These properties can be classified into *atomic*

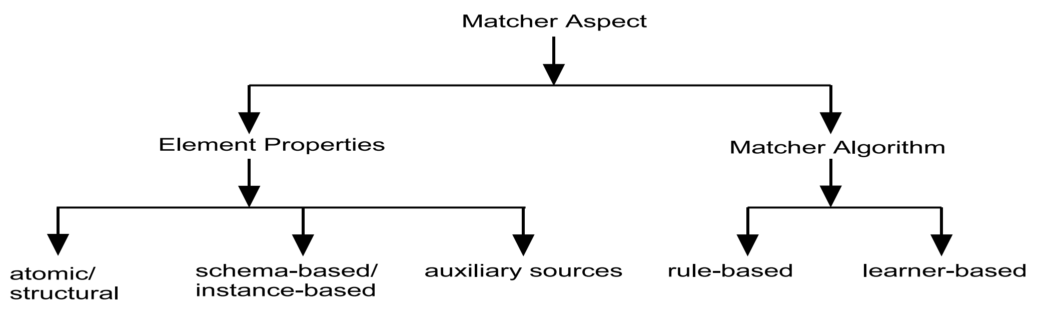


Figure 4.3: Element matcher aspects

or *structural*; *schema-based* or *instance-based*; and *auxiliary* properties. Atomic properties are the internal schema element features, such as element name, data type, and cardinality constraints. The structural properties are the combinations of element properties that appear together in a schema. These properties take into account the element position (context) in the schema graph (or schema tree). The context of an element is reflected by its descendants, ancestors, and siblings.

Schema-based properties are the properties existing in the schema itself, such as the name and data type of an attribute in relational databases. Element description, relationship types and integrity constraints are good schema-based properties. Instance-based properties are the properties that belong to the instances of the schema, like the data values stored in the database for that attribute. The instance-level properties provide important insight into the contents and meaning of schema elements. Finally, auxiliary sources can be used to gather additional element properties. As stated, a few information sources carry semantics of schema elements; both schema-based and instance-based clues are incomplete; and the matching process is subjective. Therefore, most matchers rely not only on the input schema but also on auxiliary information such as external dictionaries, previous mappings and user feedback.

- *Matcher algorithm;* To assess the element similarity, a matching algorithm should be used exploiting the element properties. A matching algorithm can either be classified by the information they exploit or by their methodologies. According to the information they exploit, as shown in Fig. 4.3, the matchers can be [117]:

  – *Individual matchers*; exploiting one type of clues (property, schema-based, instance-based, etc), or

- *Combining matchers*; exploiting multiple types of clues. They can either be *hybrid matchers* that integrate multiple criteria [93, 96, 47, 48], or *composite matchers* that combine results of independently executed matchers [50, 53, 145].

According to their methodologies, matching algorithms can be classified as either *rule-based* or *learner-based* [51]. Rule-based systems exploit only schema-based properties in a hard-coded fashion. They are easy to implement and do not need to be trained before they are put in use, and they are fast, since operate only on schema-based properties. However, rules cannot exploit instance-based properties efficiently and they cannot exploit previous match results to improve the current matching process. Motivated by the drawbacks of rule-based systems, a collection of learning-based solutions has been proposed. The learner-based systems utilize both schema-based and instance-based information. These systems depend largely on the previous phase (a training phase), since in these systems, a set of learners needs to be well-trained first. The advantage of these approaches is that they can empirically learn the similarities among data based on their instance values. The disadvantage of using learner-based approaches is that instance data is generally available in very vast quantity. Hence, the computational cost is very expensive.

We are now in a position to report on mainstream matching systems with respect to the element matcher aspects. The Cupid system [96] exploits both atomic and structural properties in two sequential steps. A linguistic matcher exploits element atomic properties, names, and data types to produce a linguistic similarity coefficient ($lsim$) between each pair of elements. The linguistic matching makes use of auxiliary information in the form of thesaurus to help match names by identifying short-forms. A structural matcher then makes use of element structural properties and produces a structure similarity ($ssim$) coefficient.

COMA (Combining matching algorithms) [47] and its extension COMA++ [21, 48] exploit different kinds of element properties and use different kinds of matching algorithms. Both utilizes atomic (simple) properties such as names, data types, structural properties, such as TypeName (data types + names), Children (child elements) and leaves (leaf elements), and auxiliary information such as synonym tables and previous mappings. The two systems utilize *simple, hybrid* and *reuse-oriented* matchers. The simple matcher depends largely on element names as well as the element data type. The hybrid matchers use a fixed combination of simple matchers and other hybrid matchers to obtain more accurate similarity values. To effectively deal with large schemas, COMA++ proposes the fragment-based matcher [118]. Following the divide and conquer technique, COMA++ decomposes a large match problem into smaller sub-problems by matching at the level of schema fragments. Both systems support two hybrid atomic-based matchers Name and TypeName, and three hybrid structural matchers NamePath, Children, and Leaves. Applying K matchers to two schemas, $S1$ with n elements and $S2$ with m elements, the output of this step is a $K \times m \times n$ cube of similarity values.

The SF system [101] uses an operator called *SFJoin* to produce the mappings between the two schema graph elements. This operator is implemented based on a fixpoint computation. As a starting point for the fixpoint computation the system uses the initalMap produced in the previous phase. The matcher element is based on the assumption that whenever two elements in the two graphs are found to be similar, the similarity of their adjacent elements increases. After a number of iterations, the initial similarity of any two nodes propagates through the graphs. The algorithm terminates after a fixpoint has been reached.

The BtreeMatch [61, 60] and PORSCHE [122] systems are two hybrid approaches to discover semantic correspondences across XML data. Both make use of atomic and structure properties of XML data elements. BTreeMatch encodes XML data as unordered labeled trees, while PORSCHE represent them as ordered labeled trees using the depth-first traversal. Both measure the label similarity between tree nodes exploiting tree element names. BTreeMatch does not utilize any external dictionary, while PORSCHE makes use of a manually defined domain-specific synonym table. To measure the structural similarity, BTreeMatch uses an index structure to improve matching performance, while PORSCHE employs a tree mining algorithm. The Bellflower system [127] uses a single element matcher that exploits element names. The matcher is implemented using a fuzzy string similarity function.

The Spicy system [35, 34] proposes an architecture to integrate schema matching and schema mapping generation. It introduces an original approach called structural analysis that uses electrical circuit representation to compare the topology and information content of data sources. The system exploits actual data instances as well as schema-based properties. it uses a mix of top-down and bottom-up search procedures to identify mappings between two schemas. It works bottom-up in the first phase by adopting a quick element-level comparison procedure to aggressively prune the search space; once the number of candidate matching structures has been restricted, it works top-down in order to select the best mappings using structural analysis. Structural analysis consists of the following steps: (1) mapping source and target subtrees into electrical circuits, (2) solving the two circuits to determine currents and voltages, (3) choosing a number of descriptive features, such as output current, average current, total and average consistency, total and average stress; assuming that each feature is computed by a function, and (4) comparing individual descriptive features.

The SemInt system [93] utilizes attribute names and field specification (schema-based properties) and data contents and statistics (instance-based properties). The system also exploits 20 different matchers; 15 metadata-based matchers and 5 instance-based matchers. It uses the properties of one database to train a network. Then, the trained network uses information extracted from the other database as input to obtain the similarity between each attribute of the second database and each category in the first one.

The LSD system [50] largely depends on instance-based properties, but at the same time it also exploits schema-based properties. Once both base and meta learners have been learned in the previous phase, LSD is ready to predict semantic mappings for new sources. To determine such mappings, LSD needs two steps. First, LSD collects all the instances of the new source elements. LSD then applies the base learners to identify the similarity. Different types of learners can be used to perform the task such as Name Matcher, Content Matcher and Naive-Bayes Learners. GLUE [54] is an extended version of LSD. Both [53] are based on machine learning techniques for individual matchers and an automatic combination of match results. However, GLUE is devoted to discover semantic matching between ontologies. QOM [62] uses both atomic and structural features (properties) of ontology entities (elements). To keep up the high quality of mapping results the system retains using as many ontology features as possible and proposes a set of similarity measures (matchers) to determine the similarity between elements.

The iMAP [44] system discovers complex matches for relational data by casting the problem of finding complex matches as search. It has three modules: *match generator, similarity estimator*, and *match selector*. During the entire matching process, these modules make use of domain knowledge and data. The match generator takes two schemas, $S1$ and $S2$, as input. For each attribute in $S2$, it produces a set of candidates which can include both 1-1 and complex matches guided by a set of search modules, such as text, numeric, and unit conversion searchers. The dual correlation mining (DCM) [77] system represents Web query interfaces as flat schemas with a set of attribute entities. The system exploits the attribute names and data type (simple properties) as well as data values (instance-based). DCM utilizes holistic schema matching that matches a set of schemas of the same domain at the same time and finds all matchings at once. To this end, it proposes a dual correlation mining through two steps. (i) Group discovery positively mines correlated attributes to form potential attribute groups. (ii) Matching discovery mines negatively correlated attribute groups to form potential $n-ary$ complex matchings.

4.4.2 Similarity combiner

The available information about semantics of schema elements may vary; at the same time, the relationships between them are fuzzy. Therefore, most of the existing matching systems make use of multiple element matchers. Every element matcher computes a similarity degree between schema element pairs exploiting a certain element property (feature). Therefore, a matching system with K element matchers produces K similarity values for every schema element pair. To combine these values into a single one, the similarity combiner is used.

The Cupid system combines the two similarity coefficients in one weighted coefficient. The weighted coefficient is the mean of $lsim$ and $ssim$: $wsim = w_{struct} \times ssim + (1 - w_{struct}) \times lsim$, where the constant w_{struct} is in the range 0 to 1. COMA/COMA++ derives combined match results from the individual matcher results stored in the similarity cube. This is achieved by what is called aggregation of matcher-specific results. The system uses four aggregation functions, namely *Max, Weighted, Average* and *Min*. Max optimistically returns the highest similarity predicted by any matcher, while Min pessimistically takes the lowest one. Average aims at compensating between the matchers and returns the average of their predicted similarities. It is a special case of Weighted, which computes a weighted sum of the similarities given a weighting scheme indicating different importance of the matchers. The result of the similarity combiner step is a *similarity matrix*.

The QOM system nonlinearly aggregates similarity values generated by different element matchers using the following function

$$sim(e_1, e_2) = \frac{\sum_{i=1...k} w_i \times adj(sim_i(e_1, e_2))}{\sum_{i=1...k} w_i} \tag{4.1}$$

where w_i is the weight for each individual similarity value, *adj* is a function to transform the original similarity value ($adj : [0, 1] \rightarrow [0, 1]$) to emphasize high individual similarities and deemphasizes low individual similarities by weighting individual similarity results with a sigmoid function first, $sim_i(e_1, e_2)$ is the individual element measure (matcher), and K is the number of used element matchers.

BTreeMatch and PORSCHE both aggregate two similarity values generated by element matchers: terminological and cosine measures for BTreeMatch and node label similarity and contextual positioning in the schema tree for PORSCHE. In the Bellfollower system, as mentioned before, a single name matcher is used. But, to build its objective function, the system needs another matcher to exploit more properties. A path length matcher is then used to capture the structural properties of schema elements. The matchers are then combined using a weighted sum. The Spicy system uses the harmonic mean to determine global similarity between two subtrees.

LSD uses a meta-learner to combine the predictions generated by the base learners. The meta-learner is trained to gain the ability to judge how well each base learner performs with respect to each element. Based on this judgment, it assigns a weight to each combination of element and base learner that indicates how much it trusts the base learner's predictions regarding the element. The system uses a technique called stacking to implement the meta-learner. SemInt does not require a similarity combiner as this step is incorporated in the element matcher step. The back-propagation neural network produces one similarity value for each element pair. The SF system also does not contain a similarity combiner as it only uses one element matcher $SFJoin$ which produces one similarity value.

iMAP uses a similarity estimator module, which computes a score for each match candidate that indicates the candidate's similarity to another element. In doing so, the similarity estimator tries to exploit additional types of information to compute a more accurate score for each match. To this end, it employs multiple evaluator modules, each of which exploits a specific type of information to suggest a score, and then combines the suggested scores into a final one. The output of this module is a matrix that stores the similarity score of <attribute, match candidate> pairs.

4.4.3 Similarity selector

At this stage, schema elements are corresponding to one or more elements from the other schema(s). The similarity selector step is concerned with selecting the most suitable mapping(s) for each element. However, even having a good ensemble of complementary element matchers cannot guarantee that an optimal mapping will always be identified as the top choice of the ensemble. To address such situations to the last possible degree, one can adapt the approach in which $\mathcal{K}$ top-ranked schema mappings are generated and examined [55]. These mappings can be ranked based on different criteria.

Cupid creates mappings by choosing pairs of schema elements with the maximal weighted similarity, i.e., $wsim$ that exceeds a predefined threshold (i.e. $wsim \geq th$). COMA/COMA++ compute the *element similarity* and the *schema similarity*. To compute the element similarity, it uses the similarity matrix to rank $S1$ correspondences in the descending order of their similarity values and then it uses one of three selecting functions. The selecting functions are: *MaxN*, constantly returning the top N candidates; *MaxDelta*, taking the candidates with the highest similarity Max as well as those with a similarity within a relative tolerance range specified by a Delta value; and *Threshold*, returning all candidates showing a similarity above a particular threshold. On the other hand, to compute the schema similarity, both systems use two strategies, namely *Average* and *Dice*.

The SF system uses an operator called *SelectThreashold* that selects the best match candidates from the list of ranked map pairs returned by the SFJoin operator. To address the selection problem, SF specifies three steps; using application-specific constraints (typing and cardinality); using selection techniques developed in matching bipartite graphs; and evaluating the usefulness of particular selection techniques and choosing the one with empirically best results. BTreeMatch selects a set of matches consisting of all element pairs whose similarity value is above a threshold given by an expert.

From the similarity values, QOM derives the actual mappings based on a threshold and a greedy strategy. QOM interprets similarity values by two means. First, it applies a threshold to discard false evidence of similarity. For the general threshold, NOM also uses a maximized f-measure of training data. Second, NOM enforces bijectivity of the mapping by ignoring candidate mappings that would violate this constraint and by favoring candidate mappings with highest aggregated similarity scores. As there may only be one best mapping, every other match is a potential mistake, which is ultimately dropped.

The OntoBuilder framework [55] introduces a technique that computes the top-$\mathcal{K}$ prefix of a consensus ranking of alternative mappings between two schemas, given the graded valid mappings of schema elements provided individually by the members of element matchers. To compute top-$\mathcal{K}$ mappings, the framework first utilizes the threshold algorithm, which requires time exponential in the size of the matched schemas. To improve matching performance, it introduces a simple algorithm, called Matrix-Direct, specific to the schema matching problem. Then, the framework makes use of both algorithms to address matching scenarios when the Matrix-Direct algorithm is not applicable.

In SemInt, users can specify filtering functions so that that the system only presents those pairs with very high similarity values. System users check and verify the output results of the trained network. LSD considers domain constraints to improve the accuracy of the mappings. The process of exploiting these constraints is called the constraint handler. The constraint handler takes the domain constraints and the prediction produced by the meta-learner to generate 1-1 mappings. The constraint handler searches through the space of possible candidate mappings to find the one with the lowest cost. Then, user feedback can further improve the mapping accuracy.

iMAP uses a match selector module that examines the similarity matrix and outputs the best matches for schema elements. The system views the task of the match selector as search for the best global match assignment that satisfies a given set of domain constraints. In doing so, it utilizes a match selector that is similar to the constraint handler module described in LSD [50]. The DCM system develops a matching construction step that consists of *matching ranking* and *matching selection*.

Table 4.1 gives a comprehensive summary for the Match phase steps w.r.t. the mentioned schema matching systems.

4.5 Mappings Transformation (MapTrans)

This phase is concerned with transforming or exporting the matching result (mappings) to be accepted by the application domain. Hence, we call it MapTrans (Mappings Transformation). As stated before, schema matching is only concerned with identifying mappings. Each mapping element reveals which elements of the two schemas are corresponding (matched) and the degree of that correspondence. But how these elements are related to each other is another related problem. This problem is known as query discovery [103, 144]. Since our main focus in this thesis is how to identify mappings but not to discover transformation rules, we only present different aspects that affect mapping transformation. The MapTrans phase inherently depends on two main aspects: *matching cardinality* and *mapping representation*.

The match cardinality is subdivided into *global* and *local* cardinality [117, 65]. Global cardinality considers how many times (zero or more) an element of a first schema $S1$ is included in a match correspondence with elements of a second schema $S2$ (i.e., an element of $S1$ can participate in zero, one, or several correspondences from $S2$ 1:1, 1:n/n:1, or n:m cardinality). While local cardinality refers to the number (one or more) of elements of $S2$ is related to within a matching correspondences (i.e., within a correspondence, one or more elements of $S1$ may be matched with one or more elements of $S2$). Most of the existing matching systems map each element of one schema to the element of the other

Table 4.1: Match phase steps w.r.t. mainstream systems

Criteria \ Prototypes	Cupid	COMA/COMA++	SemInt	LSD/GLUE	SF	iMAP	QOM	Bellflower	BTreeMatch	BORSCHE	Spicy
Element Property											
atomic	name data type	name, data type	name length	name	name	name	name	name	name	name	x
structure	leaves	node-context child, leaves, parent,	x	x	neighbors	x	hierarchical features	total path length	node context	node scope	x
schema-based	√	√	√	√	√	√	√	√	√	√	x
instance-based	x	x	√ data values& statistics	√ instances	x	√ instances	√ instances	x	x	x	√ instances
auxiliary sources	√ thesaurus	√ synonymy table	x	√ synonyms& constraints	√ specific constraints	√ domain knowledge	x	x	x	√ synonym tables	x
Element Matcher											
rule-base	√ hybrid (linguistic& structure)	√ combined (simple, hybrid,reused)	x	x	√ structure (fixpoint computation)	x	√ string similarity set/object similarity	√ name matcher structure matcher	√ string matcher cosine vector similarity	√ label matcher tree mining	√ structural analysis (electrical circuit analysis)
learner-based	x	x	√ neural network (15 meta-data & 5 data-based matchers)	√ machine learning multiple learning strategy (base& meta learners)	x	√ machine learning match generator (beam search)	x	x	x	x	x
matcher output	linguistic similarity,*lsim* structural similarity,*ssim*	k×m×n similarity cube		multi- mapping	multi- mapping	a cube of similarity	a cube of similarity	name similarity structure similarity	string similarity cosine similarity	multi-mapping	multi-mapping
Similarity Combiner											
method	√ weighted sum	√ aggregate function (max, min, avg.)	x	√ meta-learner stacking	x	√ similarity estimator (multiple evaluation modules)	√ sigmoid, sum	√ aggregate function (avg., sum, max)	√ aggregate function (avg., sum, max)	√ aggregate function (avg., sum, max)	√ harmonic mean
output	weighted sum similarity (*wsim*)	m ×n matrix similarity,	x	similarity matrix	x	similarity matrix	similarity matrix	similarity matrix	similarity matrix	multiple mappings	similarity matrix
Similarity Selector											
method	maximum weighted sum ($wsim_{max}$)	element similarity, schema similarity (maxN, maxDelta, threshold)	user specified	constraint handler/ relaxation labeler	filters (constraints, selection threshold functions(exact, best, left))	match selector (best global match)	threshold selection, bijectivity of mappings (threshold selection)	branch & bound algorithm	threshold selection	threshold selection	threshold selection
output	simple 1:1 & n:1	simple 1:1 & 1:n	simple 1:1	simple 1:1	simple 1:1	simple m:n	complex	simple 1:1	simple 1:1	simple 1:1& 1:n	simple 1:1

schema with the highest similarity. This results in local 1:1 matches and global 1:1 or 1:n mappings. Hence, the mapping transformation for these types of cardinality is a plain process. However, more work is needed to explore more sophisticated transformations coping with local and global n:1 and n:m mappings.

Another view of matching cardinality can be found in [145]. The authors define semantic correspondences between a source schema S and a target schema schema T as a set of mapping elements. A mapping element is either a *direct* or *indirect* match. A direct match binds a schema element in the source schema to a schema element in the target schema, while an indirect match binds a virtual schema element in a virtual source schema to a target schema element in the target schema through an appropriate mapping expression over the source schema. A mapping expression specifies how to derive a virtual element through manipulation operations over a source schema.

Cupid and LSD systems output element-level mappings of 1:1 local and n:1 global cardinality, where Cupid represents mappings as paths while LSD represents it as nodes. COMA and SF systems produce element-level correspondences of 1:1 local and m:n global cardinality where COMA represents mappings as paths while SF represents mappings as nodes. The SemInt system returns element-level correspondences of m:n local and global cardinality and represents the mapping elements as nodes. The element-level correspondences in these systems are associated with a similarity value in the range [0,1].

Mapping representation, the second aspect that affects mapping transformation, is primarily based on the internal representation of input schemas. Two models have been used: graphs and trees. Hence, schema elements may either be represented by nodes or paths in the schema graphs/trees which also impacts the representation of schema matches. Table 4.2 gives a summary of the schema matching phases addressed.

4.6 Summary

In this chapter, we introduced a generic schema matching framework in order to provide a base for discussing and reporting mainstream matching systems. Although each system innovates on a particular aspect, these systems have common features. In the following, we summarize some global observations concerning the presented systems:

- *Input schema;* A number of existing systems restrict themselves to accept only one data model, such as relational in SemInt and XML schemas in PORSCHE. However, a small number of systems deals with different data models, such as COMA++, and

Table 4.2: Schema Matching Phases w.r.t. mainstream systems

Criteria \ Prototypes	Cupid	COMA/COMA++	SemInt	LSD/GLUE	SF	iMAP	QOM	Bellflower	BTreeMatch	BORSCHE	Spicy
General information											
place	Washington Uni. Leipzig & Microsoft	Leipzig Uni.	Northwestern Uni.	Washington Uni.	Stanford Uni. Leipzig Uni.	Illinois Uni. Washington Uni.	Karlsruhe Uni.	Twente Uni.	Montpellier Uni. II	Montpellier Uni. II	Universit della Basilicata, Potenza
year	2001	2002/2005	2000	2001/2002	2002	2004	2004	2006	2007	2007/2008	2006/2008
TransMat											
input schema	Relational & XML	XML (xsd, xdr) & Ontology (owl, rdf)	Relational	XML /ontology	XML &Relational	relational	ontology	XML	XML	XML	XML & relational
internal representation	tree	rooted directed graph	-	tree	graph	-	tree	rooted graph (constraint problem)	tree	tree	tree (electrical circuit)
Pre-Match											
rule/learner	rule	rule	learner	learner	rule	learner	rule	rule	rule	rule	rule
process	tree traversal element identification synonym& hypernym thesaurus construction	graph traversal element identification building external dictionaries	neural network, SOM, back-propagation	machine-learning Naive Bayes	string compare	data cleaning specify domain constraints domain knowledge& data	feature engineering search select	graph traversal element identification	tree traversal element identification	tree traversal element identification abbreviation table& domain-specific synonym table	sampling circuit mapping function
output	nodes paths	nodes, paths, fragment	categories, trained network	trained base learners	initial mapping		nodes, paths instances	variables, domains constraints, objective functions	nodes paths	nodes, paths subtrees	electrical circuits
MapTrans											
match result	simple	simple	simple	simple	simple	complex	simple	simple	simple	simple	simple
mapping cardinality	1:1 & n:1	1:1 & 1:n	1:1	1:1	1:1	m:n		1:1	1:1	1:1& 1:n	1:1

Spicy.

- *Internal representation;* Some existing matching systems encode input schemas as tree-like structure, while a small number represents them as graphs.
- *Exploited information;* Some systems exploit only schema-based information, such as COMA/COMA++, SF, and Cupid, while others make advantage of both schema-based and instance-based information, like SemInt, LSD, iMAP, and Spicy.
- *Sort of matchers;* Based on the exploited information, existing matching systems can be either *rule-based* or *learner-based* systems.
- *Matching result;* Most of the existing systems convey themselves to discover simple mapping (one-to-one correspondences), such as Cupid, SF, PORSCHE, while only a small number of systems has strived to address the problem of discovering complex mappings, such as iMAP and DCM.

Part III

Design, architecture, and implementation

5 XML element similarity measures

The proliferation of shared XML-based applications necessitates the need for developing high-performance techniques to discover correspondences across large XML data efficiently. One of the most important components in XML schema matching is the definition, adoption, and utilization of element similarity measures. In this chapter, we introduce and classify several methods of element similarity measures. In particular, we first present basic concepts and definitions used through the chapter. Then, we distinguish between the internal and external element similarity measures that will be used to quantify the relationship between XML elements.

The material presented in this chapter has been developed and published in [12, 17, 5].

5.1 Preliminaries

XML is a flexible representation language. It allows to use user-defined tags without any structural constraints. As a result, XML is widely used for the representation and exchange among Web applications, and the need for developing high-performance techniques to identify and discover semantic correspondences among large XML data efficiently has been growing. The most important components to achieve this task are the definition, adoption, and utilization of element similarity measures.

In general, there are two kinds of XML data: *XML documents* and *XML schemas*. An XML schema provides the definitions and structure for XML documents. Several XML schema languages have been proposed to describe the structure and the legal building blocks in XML documents [88]. Among them, XML Document Type Definition (DTD) and XML Schema Definition (XSD) are commonly used.

XML DTD was considered the de facto standard XML schema language until the arrival of XML XSD. Its main building block consists of *element* and *attribute*. It has limited capabilities compared to XSD. XSD aims to be more expressive than DTD and more usable by a wider variety of applications. It has many novel mechanisms such as simple and complex types, rich datatype sets, occurrence constraints, inheritance, etc. Due to the wider popularity and usage of XSD [88], as mentioned in Chapter 2, we only consider XSD schemas.

An XML schema can be modeled as a graph. It can also be represented as a tree by dealing with nesting and repetition problems using a set of predefined transformation rules [89]. XML schemas are more commonly modeled as trees due to the complexity involved in graph handling algorithms, such as subgraph isomorphism. Subgraph Isomorphism, an NP-complete problem, is the problem of identifying whether a given graph, G is an isomorph of subgraph, H or not. Consequently, in the following, we represent XML schemas as rooted, labeled trees, called *schema trees*, defined as follows:

Definition 5.1 *A schema tree (ST) is a rooted, labeled tree defined as a 3-tuple $ST = \{N_T, E_T, Lab_N\}$, where:*

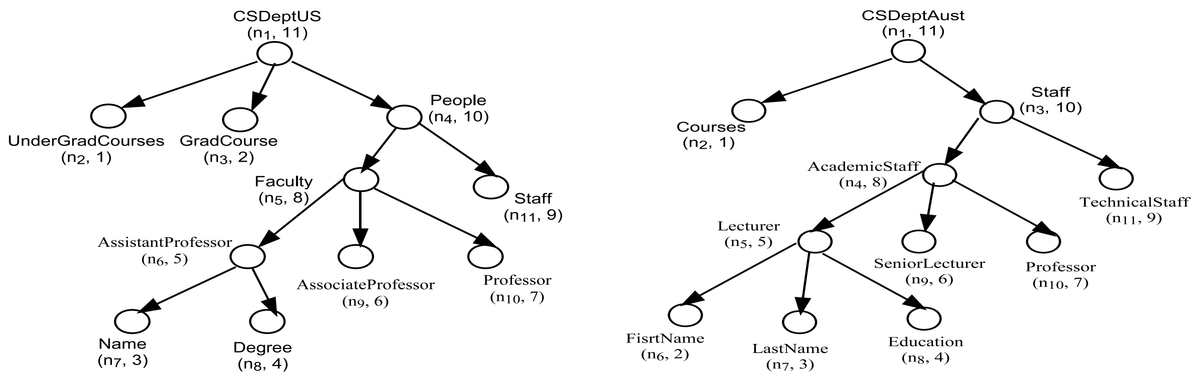
- $N_T = \{n_{root}, n_2, ..., n_n\}$ *is a finite set of nodes, each of them is uniquely identified by an object identifier (OID), where n_{root} is the tree root node. There are basically two types of nodes in a schema tree:*

  1. *Element nodes. These correspond to element declarations or complex type definitions.*
  2. *Attribute nodes. These correspond to attribute declarations.*

- $E_T = \{(n_i, n_j) | n_i, n_j \in N_T\}$ *is a finite set of edges, where n_i is the parent of n_j. Each edge represents the relationship between two nodes.*

- Lab_N *is a finite set of node labels. These labels are strings for describing the properties of the element and attribute nodes, such as name, data type, and cardinality constraint.*

A schema tree, ST, is called an ordered labeled tree if a left-to-right order among siblings in ST is given; otherwise it is called unordered schema tree. To compactly represent ordering between schema tree nodes, it is desirable to use the numbering scheme. The



(a) Schema Tree ST1 (b) Schema Tree ST2

Figure 5.1: Tree representation of XML schemas.

work of Dietz [45] is the original work on numbering schemes for trees. It labels each node in a schema tree with a pair of numbers, *(pre,post)*, which corresponds to the preorder and postorder traversal numbers of the node in the tree. In our design, without loss of generality, we choose to use the postorder traversal to uniquely number tree nodes.

Example 5.1 *To describe the operations of our study, we use the example found in [53] that has been widely used in the literature. It describes two XML schemas that represent the organization in universities from different countries. Figures 5.1(a,b) show the schema trees of the two XML schemas, wherein each node is associated by its name label, such as "CSDeptUS", its OID such as n_1, and its corresponding postorder traversal number.*

Definition 5.2 *An Element ($\mathcal{E}\ell$) is a singular data item that is the basis of the similarity measures. In a schema tree, it may be: an element node or an attribute node.*

We categorize schema tree elements into:

- *Atomic elements.* These represent simple element or attribute nodes, which have no outgoing edges and represent leaf nodes in the schema tree, and
- *Complex elements.* These represent complex element nodes, which are the internal nodes in the schema tree.

Example 5.2 *Schema tree, $ST1$, elements having OIDs $n_2, n_3, n_7, n_8, n_9, n_{10}, n_{11}$ are atomic elements, while elements having OIDs n_1, n_4, n_5, n_6 are complex elements, as shown in Fig. 5.1.*

Furthermore, there exist several relationships among schema tree elements that reflect the hierarchical nature of the XML schema tree. These relationships include:

- *parent-child (induced)* relationship, which is the relationship between each element node and its direct subelement/attribute node;
- *ancestor-descendant (embedded)* relationship, which is the relationship between each element node and its direct or indirect subelement/attribute nodes;
- *order* relationship among siblings.

Definition 5.3 *The label set associated to each element (node) in a schema tree is called the element features. Each has an associated value.*

In the XML schema context, *names, types*, and *cardinality constraints* are the most commonly used properties (features) for XML elements and attributes. Consequently, we also make use of these features when assessing the similarity between schema elements. Figure 5.1 shows that each schema tree element has a name feature represented by its value. We represent this using the dot (.) notation, such as $ST1.n_2.name$ = "*Grad-Course*" and $ST2.n3.name$ = "*Staff*".

Definition 5.4 *A similarity measure* $Sim : ST1 \times ST2 \rightarrow \mathbb{R}$*, is a metric used to quantify the similarity between elements* $\mathcal{E}\ell_1 \in ST1$ *and* $\mathcal{E}\ell_2 \in ST2$ *such that:*

$$\forall \mathcal{E}\ell_1 \in ST1, \mathcal{E}\ell_2 \in ST2, Sim(\mathcal{E}\ell_1, \mathcal{E}\ell_2) \geq 0$$

$$\forall \mathcal{E}\ell \in ST, Sim(\mathcal{E}\ell, \mathcal{E}\ell) = 1$$

$$\forall \mathcal{E}\ell_1 \in ST1, \mathcal{E}\ell_2 \in ST2, Sim(\mathcal{E}\ell_1, \mathcal{E}\ell_2) = Sim(\mathcal{E}\ell_2, \mathcal{E}\ell_1)$$

A similarity measure exploits the features of elements as well as the relationships among them to determine the similarity between a pair of elements. It is represented as $Sim(\mathcal{E}\ell_1, \mathcal{E}\ell_2)$, and its value is computed by the employed method. Usually, the similarity value ranges between 0 and 1, when the measure is normalized. The value of 0 means strong dissimilarity between elements, while the value of 1 means exact same elements. The similarity between two elements $\mathcal{E}\ell_1 \in ST1, \mathcal{E}\ell_2 \in ST2$ can be determined using the following equation:

$$Sim(\mathcal{E}\ell_1, \mathcal{E}\ell_2) = Combine(InterSim(\mathcal{E}\ell_1, \mathcal{E}\ell_2), ExterSim(\mathcal{E}\ell_1, \mathcal{E}\ell_2)) \quad (5.1)$$

where $InterSim(\mathcal{E}\ell_1, \mathcal{E}\ell_2)$ represents the internal similarity measure between two elements exploiting their features, while $ExterSim(\mathcal{E}\ell_1, \mathcal{E}\ell_2)$ represents the external similarity measure between them exploiting their hierarchal relationships, and and $Combine$ is an aggregation function used to combine and quantify the importance of $InterSim$ measure and $ExterSim$ measure.

In the following, we present different techniques used to determine the internal and external similarity between schema tree elements.

5.2 Internal element similarity measures

The internal element measures exploit their own features, such as their *names*, *data types*, *constraints*, etc. to compare elements from different schema trees. Depending on the type of exploited feature, we present the following internal measures.

5.2.1 Name similarity measure

In the absence of data instances, the element name is considered an important source of semantic information for schema matching [96]. Element names can be syntactically similar *(Staff, TechnicalStaff)* or semantically similar *(People, Staff)*. As a result, it is desirable to consider both *syntactic* and *semantic* measures to compute the degree of similarity between element names. In order to make element names comparable, they should be normalized into a set of tokens. The normalization process may have the following steps:

- *Tokenization*: The element name is parsed into a set of tokens using delimiters, such as punctuation, uppercase or special symbols, etc. E.g. $UnderGradCourses \rightarrow \{Under, Grad, Courses\}$.

- *Expansion*: Abbreviations and acronyms are expanded. E.g. $Grad \rightarrow Graduate$.

- *Elimination*: Tokens that are neither letters nor digits are eliminated and ignored during the comparison process.

After decomposing each element name into a set of tokens, the name similarity between the two sets of name tokens T_1 and T_2 is determined as the average best similarity of each token with all tokens in the other set [96, 108]. It is computed as follow:

$$Nsim(T1, T2) = \frac{\sum_{t_1 \in T_1}[\max_{t_2 \in T_2} sim(t_1, t_2)] + \sum_{t_2 \in T_2}[\max_{t_1 \in T_1} sim(t_2, t_1)]}{|T1| + |T2|} \qquad (5.2)$$

To determine the similarity between a pair of tokens, $sim(t_1, t_2)$, both syntactic and semantic measures can be used.

Syntactic measures (String-based)

Syntactic measures take the advantage of the representation of element names as strings (sequence of characters). There are many methods to compare strings depending on the way the string is seen (as exact sequence of characters, an erroneous sequence of characters, a set of characters,..) [105, 41, 63].

1. **Edit distance**. The edit distance between two strings is the number of operations required to transform one string into the other. There are several algorithms to define or calculate this metric. In our implementation we make use of the following:

   - *Levenshtein distance.* The Levenshtein distance between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. To compute the similarity degree between two tokens $t_1 \in T_1$ and $t_2 \in T_2$, the following equation is used

     $$sim_{edit}(t_1, t_2) = \frac{max(|t_1|, |t_2|) - editDistance(t_1, t_2)}{max(|t_1|, |t_2|)} \qquad (5.3)$$

     where *editDistance(t_1, t_2)* is the minimum number of character insertion, deletion, and substitution operations needed to transform t_1 to t_2. Each edit operation is assigned a unit cost.

   - *Jaro Similarity.* The Jaro measure of similarity between two strings is mainly used in the area of record linkage (duplicate detection). The higher the Jaro measure for two strings is, the more similar the strings are. The Jaro measure is designed and best suited for short strings such as person names. The score is normalized such that 0 equates to no similarity and 1 is an exact match. The

Jaro similarity measure is calculated as [41]:

$$sim_{jaro}(t_1, t_2) = \frac{1}{3} \times (\frac{M}{|t_1|} + \frac{M}{|t_2|} - \frac{M-t}{M}) \tag{5.4}$$

where M is the number of matching characters and t is the number of transpositions. A variant of this measure from Winkler uses the length P of the longest common prefix of the two string. Let $P' = max(P, 4)$, the Jaro-Winkler measure is defined as [41]

$$sim_{jaro-winkler}(t_1, t_2) = sim_{jaro}(t_1, t_2) + \frac{P'}{10} \times (1 - sim_{jaro}(t_1, t_2)) \tag{5.5}$$

2. **N-gram distance**. N-grams are typically used in approximate string matching by sliding a window of length n over the characters of a string to create a number of 'n' length grams for finding a match. The 'n' length grams are then rated as number of n-gram matches within the second string over possible n-grams. An n-gram of size 1 is referred to as a "uni-gram"; size 2 is a "di-gram"; size 3 is a "tri-gram"; and size 4 or more is simply called n-gram. The intuition behind the use of n-grams as a foundation for approximate string processing is that when two strings t_1 and t_2 are within a small edit distance of each other, they share a large number of n-grams in common. The n-gram between two string t_1 and t_2 is defined as

$$sim_{n-gram}(t_1, t_2) = \frac{2 \times |n - gram(t_1) \bigcap n - gram(t_2)|}{|n - gram(t_1)| + |n - gram(t_2)|} \tag{5.6}$$

where *n-gram(t)* is the set of n-grams in t. In our implementation, we make use of the tri-gram measure. Hence, the above equation can be rewritten as follows

$$sim_{tri}(t_1, t_2) = \frac{2 \times |tri(t_1) \bigcap tri(t_2)|}{|tri(t_1)| + |tri(t_2)|} \tag{5.7}$$

where $tri(t_1)$ is the set of trigrams in t_1.

Example 5.3 *Table 5.1 represents different string-based similarity measures used to compute the degree of similarity between the pair of element names, "UnderGradCourses"* [1] *and "Courses".*

[1]only tokenization is considered without expansion

Table 5.1: Example of using string-based measures.

T_1	T_2	$sim_{edit}(t_1, t_2)$	$sim_{Jaro}(t_1, t_2)$	$sim_{tri-gram}(t_1, t_2)$	$sim_{syn1}(t_1, t_2)$ (using the average)	$sim_{syn2}(t_1, t_2)$ (using weighted sum)
under	course	0.2857	0.565	0	0.2836	0.1695
grad	course	0.142	0.464	0	0.202	0.196
courses	courses	1.0	1.0	1.0	1.0	1.0
$Nsim(T_1, T_2)$		0.582	0.752	0.5	0.6214	0.5913

Semantic measures (Language-based)

The semantic measures are based on using Natural Language Processing (NLP) techniques to find the degree of similarity between schema tree element names. Most of these techniques rely heavily on the use of external sources, such as dictionaries and lexicons. Typically, WordNet is used either to simply find close relationships, such as synonyms between element names, or to compute some kind of semantic distance between them. The sMatch system [70] proposes a semantic schema matching exploiting the features in WordNet as a background knowledge source to return semantic relations (e.g., equivalence, more general) between element names rather than similarity values in the [0,1] range. Another possibility is to utilize a domain-specific user-defined dictionary. COMA++ [48] and PORSCHE [122] utilize a user-defined dictionary to get a similarity degree between element names.

Invoking the external sources, such as WordNet, to determine the semantic similarity between element names makes the matching process slow, especially in the large-scale context. As our main aim is to develop a schema matching approach that copes with large-scale XML data, we do not make use of any external dictionaries or ontologies. This means that to assess the similarity between element names, we rely only on string-based measures.

5.2.2 Data type similarity measure

Although the element name is considered a necessary source for determining the element similarity, however, it is an insufficient source. For example, the name similarity between two elements $ST1.n_9$ and $ST2.n_3$, see Fig. 5.1, equals 1.0. This is a false positive match, as these two elements are of different data types. This necessitates the need for other schema information sources used to compute the element similarity and to prune some of these false positive matches. The element data type is another schema information source that makes a contribution in determining the element similarity.

Type1	Type2	Tsim
string	string	1.0
string	decimal	0.2
decimal	float	0.8
float	float	1.0
float	integer	0.8
integer	short	0.8
...	...	...

Figure 5.2: Data type similarity table.

XML schema supports 44 primitive and derived built-in data types[2]. Using the XML built-in data type hierarchy, a data type similarity can be computed. One method is to build a data type similarity table used in [96, 108]. Figure 5.2 illustrates that elements having the same data types or belonging to the same data type category have the possibility to be similar and their type similarities (Tsim) are high. For elements having the same data types, the type similarity is set to 1.0, while the type similarity of elements having different data types but belonging to the same category (such as integer and short) is set to 0.8.

Example 5.4 *Fig. 5.1 depicts the element $ST_1.n_9$ having a string data type (atomic element), and the element $ST_2.n_3$ having a complex type (complex element). $Tsim(string, complex) = 0$, which increases the possibility that the two elements are not similar.*

5.2.3 Constraint similarity measure

Another schema information source of the element that makes another contribution in assessing the element similarity are its constraints. The cardinality (occurrence) constraint is considered the most significant. The *minOccurs* and *maxOccurs* in the XML schema define the minimum and maximum occurrence of an element that may appear in XML documents. A cardinality table for DTD constraints has been proposed in [89]. The authors of [108] adapt this table for constraint similarity of XML schemas. Figure 5.3 shows the cardinality constraint similarity, where ”none” is equivalent to minOccrs=1 and maxOccurs=1, “?” is equivalent to minOccurs=0 and maxOccurs=1, “*” is equivalent to minOccurs=0 and maxOccur=unbounded, and “+” is equivalent to minOccurs=1 and maxOccurs=unbounded.

[2]`http://www.w3.org/TR/xmlschema-2/`

	*	+	?	none
*	1	0.9	0.7	0.7
+	0.9	1	0.7	0.7
?	0.7	0.7	1	0.8
none	0.7	0.7	0.8	1

Figure 5.3: Cardinality constraint similarity table.

Putting all together

In the schema matching context, a few matching systems use only one of the internal measures to produce an initial matching result, such as the Similarity Flooding system [101], which uses a simple string matcher that compares common prefixes and suffixes of literals. However, the majority of these systems make use of multiple string-based methods, such as COMA++ [48] and BtreeMatch [61], which utilize Edit distance and n-gram as name similarity measures. Moreover, COMA++ makes use of element data types. In this situation, the arising question is how to combine these different measures [55, 90]. Most of the existing matching systems use aggregation functions, such as the *weighted sum*, the *average* functions, etc. to combine different similarity measures. In our implementation, we use the weighted sum as the combination strategy.

Definition 5.5 *The internal similarity of two elements $\mathcal{E}\ell_1 \in ST_1$ and $\mathcal{E}\ell_2 \in ST_2$ is the combination of the name similarity ($Nsim$), data type similarity ($Tsim$), and constraint similarity ($Csim$):*

$$\begin{aligned} InterSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right) &= Combine_I \quad \left(Nsim\left(\mathcal{E}\ell_1.name, \mathcal{E}\ell_2.name\right),\right. \\ &\qquad\qquad Tsim\left(\mathcal{E}\ell_1.type, \mathcal{E}\ell_2.type\right), \\ &\qquad\qquad \left.Csim\left(\mathcal{E}\ell_1.card, \mathcal{E}\ell_2.card\right)\right) \end{aligned}$$

where $Combine_I$ is an aggregation function used to combine the three similarity measures.

5.3 External element similarity measures

In contrast to internal element measures that exploit the element features without considering the position (context) of the element. They do not consider the impact of other

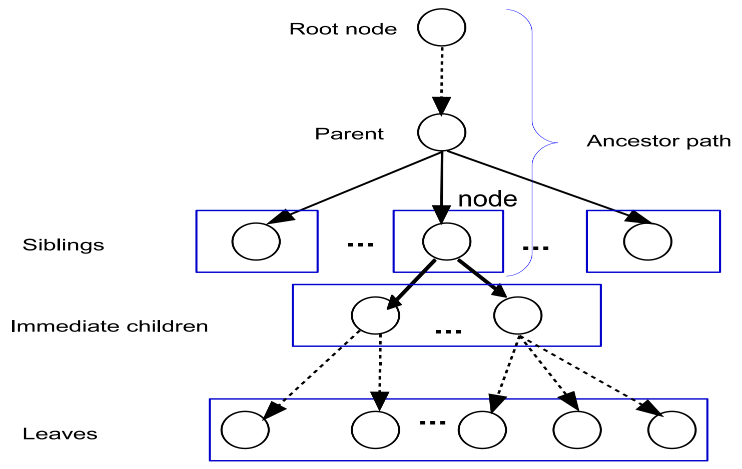


Figure 5.4: The context of an XML tree element.

surrounding elements. To consider the context of the element, the external measures make use of the element relationships instead of its features.

5.3.1 Element context

The external element measures rely heavily on the element context, as shown in Fig. 5.4, which is reflected by its descendants, ancestors, and siblings. The descendants of an element include both its immediate children and the leaves of the subtrees rooted at the element. The immediate children reflect its basic structure, while the leaves reflect the element's content. In our implementation, we utilize the following element (node) contexts:

- *The child context* of an element is defined as the set of its immediate children nodes including attributes and subelements. The child context of an atomic element (leaf node) is an empty set.

- *The leaf context* of an element is defined as the set of leaf nodes of subtrees rooted at the element. The leaf context of a leaf node is an empty set.

- *The ancestor context* of an element is defined as the path extending from the root node to the element. The ancestor context of the root node is an empty path.

- *The sibling context* of an element contains both the preceding siblings and the following siblings of the element.

To extract the context of a schema tree element, $\mathcal{E}\ell$, we respectively define the following four functions:

- $child(\mathcal{E}\ell) = \{\mathcal{E}\ell_i |\ (\mathcal{E}\ell, \mathcal{E}\ell_i) \in E_T\}$; is a function that returns the child context set for the element.
- $leaf(\mathcal{E}\ell) = \{\mathcal{E}\ell_i |\ \mathcal{E}\ell_i\, is\, an\, atomic\, element \wedge there\, is\, a\, path\, between\, \mathcal{E}\ell_i\, and\, \mathcal{E}\ell\}$; is a function that determines the leaf context set of the element.
- $ancestor(\mathcal{E}\ell) = \mathcal{E}\ell_{root}/.../\mathcal{E}\ell$; is a function that identifies the ancestor context path for the element.
- $sibling(\mathcal{E}\ell) = \{\mathcal{E}\ell_i |\ \exists \mathcal{E}\ell_j s.t.\ (\mathcal{E}\ell_j, \mathcal{E}\ell) \in E_T \wedge (\mathcal{E}\ell_j, \mathcal{E}\ell_i) \in E_T\}$; is a function that returns the sibling context set of the element.

Example 5.5 *Consider schema trees shown in Figure 5.1,* $child(ST1.n_1) = \{n_2, n_3, n_4\}$, $leaf(ST2.n_4) = \{n_6, n_7, n_8, n_9, n_{10}\}$, $ancestor(ST1.n_9) = n_1/n_4/n_5/n_9$, *and* $sibling(ST2.n_7) = \{n_6, n_8\}$

5.3.2 Element context measure

The context of an element is the combination of its *child*, *leaf*, *ancestor*, and *sibling* contexts. Two elements are structurally similar if they have similar contexts. The structural node context defined above relies on the notions of *path* and *set*. In order to compare two ancestor contexts, we essentially compare their corresponding paths. On the other hand, in order to compare two child contexts, leaf, and/or sibling contexts, we need to compare their corresponding sets.

Although path comparison has been widely used in XML query answering frameworks, it relies on strong matching following the two crisp constraints: node constraint and edge constraint. Under such constraints, paths that are semantically similar may be considered as unmatched, or paths that are not semantically similar may be considered as matched. Hence, these constraints should be relaxed. Several relaxation approaches have been proposed to approximate answering of queries [18]. Examples of such relaxations are allowing matching paths even when nodes are not embedded in the same manner or in the same order, and allowing two elements within each path to be matched even if they are not identical but their linguistic similarity exceeds a fixed threshold [37].

To determine the context (structural) similarity between two elements $\mathcal{E}\ell_1 \in ST_1$ and $\mathcal{E}\ell_2 \in ST_2$, the similarity of their child, leaf, ancestor, and sibling contexts should be computed.

1. *Child context similarity.* The child context set is first extracted for each element, say $C\_set_1 = child(\mathcal{E}\ell_1) = \{\mathcal{E}\ell_{11}, \mathcal{E}\ell_{12}, ..., \mathcal{E}\ell_{1k}\}$ and $C\_set_2 = child(\mathcal{E}\ell_2) = \{\mathcal{E}\ell_{21}, \mathcal{E}\ell_{22}, ..., \mathcal{E}\ell_{2k'}\}$. The internal similarity between each pair of children in the two sets is then determined, the matching pairs with maximum similarity values are selected, and finally the average of best similarity values is computed. The child context similarity, $ChSim$, can be computed using

$$ChSim(\mathcal{E}\ell_1, \mathcal{E}\ell_2) = \frac{\sum_{i=1}^{i=k}[\max_{j=1}^{j=k'} InterSim(\mathcal{E}\ell_{1i}, \mathcal{E}\ell_{2j})]}{max(|k|, |k'|)} \qquad \boxed{5.8}$$

   where $InterSim(\mathcal{E}\ell_{1i}, \mathcal{E}\ell_{2j})$ is the internal similarity computed using Definition 5.5.

2. *Leaf context similarity.* Before getting into the details of computing the leaf context similarity, we first introduce the notion of *gap* between two elements and the gap vector of an element.

   Definition 5.6 *The gap between two elements $\mathcal{E}\ell_i$ and $\mathcal{E}\ell_j$ in a schema tree ST is defined as the difference between their postorder numbers.*

   Definition 5.7 *The gaps between a complex element and its leaf set in a schema tree form a vector called the gap vector.*

   Example 5.6 *Considering $ST1$ of Example 5.1, the gap vector (gapvec) of node n_5 is $gapvec(n_5) = \{5, 4, 2, 1\}$.*

   To compute the leaf context similarity between two elements, we compare their leaf context sets. To this purpose, first, we extract the leaf context set for each element, say $L\_set_1 = leaf(\mathcal{E}\ell_1) = \{\mathcal{E}\ell_{11}, \mathcal{E}\ell_{12}, ..., \mathcal{E}\ell_{1k}\}$ and $L\_set_2 = leaf(\mathcal{E}\ell_2) = \{\mathcal{E}\ell_{21}, \mathcal{E}\ell_{22}, ..., \mathcal{E}\ell_{2k'}\}$. Then, we determine the gap vector *(gapvec)* for each element, say v_1 and v_2. We finally apply the cosine measure (CM) between the two vectors. The cosine measure between the two gap vectors is given by the following formula

$$CM(v_1, v_2) = \frac{v_1 v_2}{|v_1||v_2|} \tag{5.9}$$

CM is in the interval [0,1]. A result close to 1 indicates that the vectors tend in the same direction, and a value close to 0 denotes a total dissimilarity between the two vectors.

Example 5.7 *Considering the two schema trees shown in Example 5.1, the leaf context set of ST1.n_1 is L_set1= leaf(n_1(11))={n_2(1), n_3(2), n_7(3), n_8(4), n_9(6), n_{10}(7), n_{11}(9)}, and the leaf context set of ST2.n_1 is L_set2 =leaf(n_1(11))={n_2(1), n_6(2), n_7(3), n_8(4), n_9(6), n_{10}(7), n_{11}(9)}. The gap vector of ST1.n_1 is v_1 =gapvec(ST1.n_1)={10,9,8,7,5,4,2}, and the gap vector of ST2.n_1 is v_2 =gapvec(ST2.n_1)={10,9,8,7,5,4,2}. The cosine measure CM of the two vectors gives CM(v_1,v_2) =1.0. Then, the leaf context similarity between nodes ST1.n_1 and ST2.n_1 is 1.0. Similarly, the leaf context similarity between ST1.n_5 and ST2.n_4 is 0.82 and ST1.n_4 and ST2.n_3 is 0.98.*

3. *Sibling context similarity.* To compute the sibling context similarity between two elements, we compare their sibling context sets. For this, first, the sibling context set is extracted for each element, say $S\_set_1 = sibling(\mathcal{E}\ell_1) = \{\mathcal{E}\ell_{11}, \mathcal{E}\ell_{12}, ..., \mathcal{E}\ell_{1k}\}$ and $S\_set_2 = sibling(\mathcal{E}\ell_2) = \{\mathcal{E}\ell_{21}, \mathcal{E}\ell_{22}, ..., \mathcal{E}\ell_{2k'}\}$. The internal similarity between each pair of siblings in the two sets is then determined, the matching pairs with maximum similarity values are selected, and finally the average of best similarity values is computed. The sibling context similarity, $SibSim$, can be computed using

$$SibSim(\mathcal{E}\ell_1, \mathcal{E}\ell_2) = \frac{\sum_{i=1}^{i=k}[\max_{j=1}^{j=k'} InterSim(\mathcal{E}\ell_{1i}, \mathcal{E}\ell_{2j})]}{max(|k|, |k'|)} \tag{5.10}$$

where $InterSim(\mathcal{E}\ell_{1i}, \mathcal{E}\ell_{2j})$ is the internal similarity computed using Definition 5.5.

4. *Ancestor context similarity.* The ancestor context similarity captures the similarity between two elements based on their ancestor contexts. As mentioned before, the ancestor context for a given element $\mathcal{E}\ell_i$ is the path extending from the root node to $\mathcal{E}\ell_i$. To compute the ancestor similarity between two elements $\mathcal{E}\ell_1 \in ST1$and $\mathcal{E}\ell_2 \in ST2$, first we extract each ancestor context for each element, say $P_1 = ancestor(\mathcal{E}\ell_1)$ and $P_2 = ancestor(\mathcal{E}\ell_2)$. Then, we compare two paths, $P1$ and $P2$

assuming $|P1| < |P2|$ and utilizing three of four scores established in [39] and reused in [37].

- *LCS(P_1,P_2)*; This score is used to ensure that path P_1 includes most of the nodes of P_2 in the right order. To this end, a classical dynamic programming algorithm is employed to compute a Longest Common Subsequence (LCS) between the two paths represented as element (node) sequences $P_1[\mathcal{E}\ell_{root1}...\mathcal{E}\ell_k]$ and $P_2[\mathcal{E}\ell_{root2}...\mathcal{E}\ell_{k'}]$. Finding the longest common subsequence is a well-defined problem in the literature [27]. The process is computed by finding the LCS lengths for all possible prefix combinations of P_1 and P_2. The computed LCS lengths are stored in a matrix. The recursive equation, *Equation 5.11*, illustrates the matrix entries, where $InterSim(\mathcal{E}\ell_i, \mathcal{E}\ell_j)$ is the internal similarity computed using Definition 5.5 that assesses the internal similarity between the two elements $\mathcal{E}\ell_i$ and $\mathcal{E}\ell_j$, and th is a predefined threshold.

$$LCSM[i,j] = \begin{cases} 0 & \text{if } i = 0, j = 0 \\ LCSM[i-1, j-1] + 1 & InterSim(\mathcal{E}\ell_i, \mathcal{E}\ell_j) \geq th \\ max(LCSM[i-1, j], LCSM[i, j-1]) & InterSim(\mathcal{E}\ell_i, \mathcal{E}\ell_j) < th \end{cases} \tag{5.11}$$

  The bottom-right corner entry $LCSM[k, k']$ contains the overall LCS length. Then, we normalize this score in [0,1] by the length of path P_1. The normalized score is given by:

$$LCS_n(P_1, P_2) = \frac{|LCS(P_1, P_2)|}{|P_1|} \tag{5.12}$$

- *GAPS(P_1,P_2)*; This measure is used to ensure that the occurrences of the P_1 nodes in P_2 are close to each other. For this, another version of the LCS algorithm is exploited in order to capture the LCS alignment with minimum gaps. We propose to normalize it by the length of the common subsequence added to the gaps value so as to ensure that the total score will be less than 1. Equation 5.13 presents the normalized measure.

$$GAPS(P_1, P_2) = \frac{gaps}{gaps + LSC(P_1, P_2)} \tag{5.13}$$

- *LD(P_1,P_2)*: Finally, in order to give higher values to source paths whose lengths are similar to target paths, we compute the length difference, *LD*, between P_2 and *LCS(*P_1*,*P_2*)* normalized by the length of P_2. Thus, the final factor that evaluates the length difference can be computed as:

$$LD(P_1, P_2) = \frac{|P_2| - LCS(P_1, P_2)}{|P_2|} \tag{5.14}$$

These scores are combined to compute the similarity between the two paths P_1 and P_2 *PSim* as follows:

$$PSim(P_1, P_2) = LCS_n(P_1, P_2) - \gamma GAPS(P_1, P_2) - \delta LD(P_1, P_2) \tag{5.15}$$

where γ and δ are positive parameters ranging from 0 to 1 that represent the comparative importance of each factor.

Example 5.8 *Considering the two schema trees shown in Example 5.1, the ancestor context path of ST1.*n_5 *is P1= ancestor(*n_5*)=*$n_1/n_4/n_5$ *= CSDeptUS/People/Faculty, and the ancestor context path of ST2.*n_{10} *is P2= ancestor(*n_{10}*)=*$n_1/n_3/n_4/n_{10}$*= CSDeptAust/Staff/AcademicStaff/Professor. We have* $|LCS(P1, P2) = 1|$, *gaps*$(P1, P2) = 0$, *and* $LD(P1, P2) = \frac{4-1}{4} = 0.75$. *Setting* $\gamma = 0.25$ *and* $\delta = 0.2$, *the path similarity between the two paths becomes* $PSim(P1, P2) = 0.18$.

5.3.3 Putting all together

To get precise structural similarities between elements, it is desirable to consider all the element contexts. In this situation, the arising conflict is how to combine these context measures. In the light of this, we are now in a position to define the external element similarity measure.

Definition 5.8 *The external similarity of two elements* $\mathcal{E}\ell_1 \in ST_1$ *and* $\mathcal{E}\ell_2 \in ST_2$ *is the combination of the child context similarity (ChSim), leaf context similarity (LeafSim), sibling context similarity (SibSim), and path similarity (PSim):*

$$\begin{aligned} ExterSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right) \; = Combine_E \; & \left(ChSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right),\right. \\ & LeafSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right), \\ & SibSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right), \\ & \left. PSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right)\right) \end{aligned}$$

where $Combine_E$ is an aggregation function used to combine the external similarity measures.

5.4 The utilization of element similarity measures

In the previous sections, we defined and adopted several measures to assess the similarity between XML schema elements. We classified these measures based on the exploited information of schema elements either on internal own features or on external relationships between elements. The arising question is how to utilize these measures in a both effective and efficient way to achieve the highest matching result.

The tree representation of XML schemas can only ensure the matching quality and cannot guarantee the matching efficiency, especially in the large-scale context. Schema trees are required to be traversed many times during the application of similarity measures. As known, the time complexity of tree-based algorithms is expensive and as a result the matching efficiency declines radically. To overcome these challenges, instead of applying similarity measures to schema trees, we represent schema trees as sequences using the Prüfer encoding method [115]. The idea of Prüfer's method is to find a one-to-one correspondence between the set of the schema trees and a set of Prüfer sequences.

Definition 5.9 *A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n with repetitions allowed.*

Example 5.9 *[142] The set of Prüfer sequences of length 2 (n = 4) is {(1,1), (1,2), (1,3), (1,4), (2,1), (2,2), (2,3), (2,4), (3,1), (3,2), (3,3), (3,4), (4,1), (4,1), (4,3), (4,4)}. In total, there are $4^{4-2} = 16$ Prüfer sequences of length 2.*

Given a schema tree with nodes labeled by $1, 2, 3, ..., n$ the Prüfer encoding method outputs a unique Prüfer sequence of length $n-2$. It initializes with an empty sequence. If the tree has more than two nodes, the algorithm finds the leaf with the lowest label, and appends the label of the parent of that leaf to the sequence. Then, the leaf with the lowest label is deleted from the tree. This operation is repeated $n-2$ times until only two nodes remain in the tree. The algorithm ends up deleting $n-2$ nodes. Therefore, the resulting sequence is of length $n-2$. Figure 5.5 illustrates the Prüfer sequence (Pr.Seq) construction for a schema tree whose nodes are randomly labeled. As shown in Fig. 5.5, since the regular Prüfer sequences include only the information of parent nodes, these sequences cannot represent the leaf nodes. In order to incorporate them, a modified version of the regular Prüfer sequence is exploited in the next section.

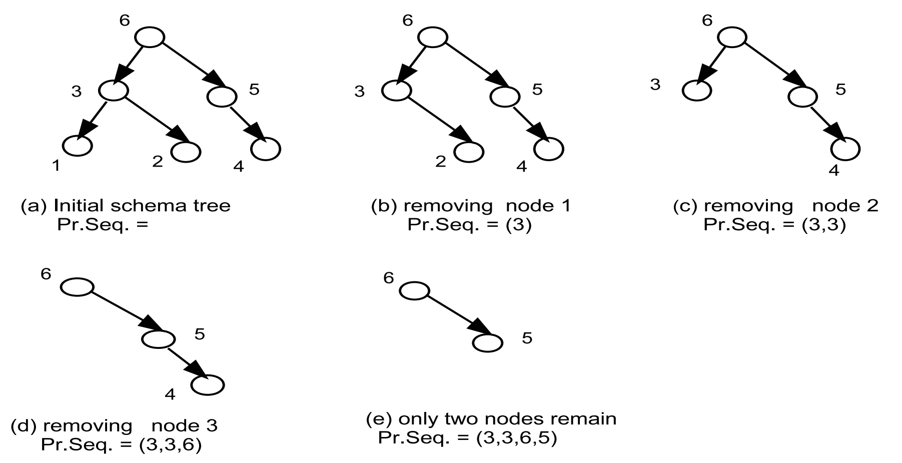


Figure 5.5: Prüfer sequence construction.

5.4.1 Prüfer sequences construction

We now describe the tree sequence representation method, which provides a bijection between ordered, labeled trees and sequences. This representation is inspired from classical *Prüfer sequences* [115] and particularly from what is called *Consolidated Prüfer Sequence, CPS*, proposed in [133].

CPS of a schema tree ST consists of two sequences, Number Prüfer Sequence *NPS* and Label Prüfer Sequence *LPS*. They are constructed by doing a *postorder traversal* that tags each node in the schema tree with a unique traversal number. NPS is then constructed iteratively by removing the node with the smallest traversal number and appending its parent node number to the already structured partial sequence. LPS is constructed similarly, but by taking the node labels of deleted nodes instead of their parent node numbers. Both NPS and LPS convey completely different but complementary information—NPS that is constructed from unique postorder traversal numbers gives tree structure information and LPS gives tree semantic information. CPS representation thus provides a bijection between ordered, labeled trees and sequences. Therefore, CPS = (NPS, LPS) uniquely represents a rooted, ordered, labeled tree, where each entry in the CPS corresponds to an edge in the schema tree. For more details see [133].

Example 5.10 *Considering schema trees ST1 and ST2 shown in Fig. 5.1, each node is associated with its OID and its postorder number. Table 5.2 illustrates CPS for ST1 and ST2. For example, CPS of ST1 can be written as the NPS(ST1)= (11 11 5 5 8 8 8 10 10 11 -), and the LPS(ST1).name= (UnderGradCourses, GradCourses, Name, Degree,* *AssistantProfessor, AssociateProfessor, Professor, Faculty, Staff, People, CSDeptUS).*

This example shows that using CPS to represent schema trees as sequences has the advantage of capturing semantic and structural information of the schema tree including atomic nodes. The following section formally presents CPS properties.

Table 5.2: Schema tree nodes properties

Schema Tree ST1						Schema Tree ST2					
NPS	*LPS*					*NPS*	*LPS*				
	OID	name	type/data type	cardinality constraint			OID	name	type/data type	cardinality constraint	
				minOccurs	maxOccurs					minOccurs	maxOccurs
11	n_2	UnderGradCourses	element/string	0	1	11	n_2	Courses	element/string	0	1
11	n_3	GradCourses	element/string	0	1	5	n_6	FirstName	element/string	0	1
5	n_7	Name	element/string	0	1	5	n_7	LastName	element/string	0	1
5	n_8	Degree	element/string	0	1	5	n_8	Education	element/string	0	1
8	n_6	AssistantProfessor	element/-	0	unbounded	8	n_5	Lecturer	element/-	0	unbounded
8	n_9	AssociateProfessor	element/string	0	1	8	n_9	SeniorLecturer	element/string	0	1
8	n_{10}	Professor	element/string	0	1	8	n_{10}	Professor	element/string	0	1
10	n_5	Faculty	element/-	0	unbounded	10	n_4	AcademicStaff	element/-	0	unbounded
10	n_{11}	Staff	element/string	0	1	10	n_{11}	TecnicalStaff	element/string	0	1
11	n_4	People	element/-	0	unbounded	11	n_3	Staff	element/-	0	unbounded
-	n_1	CSDeptUS	element/-	0	unbounded	-	n_1	CSDeptAust	element/-	0	unbounded

CPS Properties

In the following, we list the structural properties behind the CPS representation of schema trees. If we construct a CPS=(NPS, LPS) from a schema tree ST, we can classify these properties into:

- Unary Properties. Let $\mathcal{E}\ell_i$ be an element having a postorder number k,
    1. *atomic element*; $\mathcal{E}\ell_i$ is an atomic element iff $k \notin NPS$
    2. *complex element*; $\mathcal{E}\ell_i$ is a complex element iff $k \in NPS$
    3. *root node*; $\mathcal{E}\ell_i$ is the root node (n_{root}) iff $k = max(NPS)$, where max is a function which returns the maximum number in NPS.
- Binary Properties
    1. *edge relationship*; Let $CPS_i = (k_i, LPS_i) \in CPS$ of a schema tree ST be an entry. This entry represents an edge from the node whose postorder number is k_i to a node $n_i = LPS_i.OID$. This property shows both child and parent relationships. This means that the node $n_i = LPS_i.OID$ is an immediate child of the node whose postorder number is k_i.

2. *sibling relationship*; Let $CPS_i = (k_i, LPS_i)$ and $CPS_j = (k_j, LPS_j)$ be two entries $\in CPS$ of a schema tree ST. The two elements $n_i = LPS_i.OID$ and $n_j = LPS_j.OID$ are two sibling nodes iff $k_i = k_j$.

Connecting between element context and CPS

After introducing the concept of Consolidated Prüfer Sequence (CPS) and listing its properties, we build a connection between the element context and the CPS sequence. To build such a connection, we make use of CPS properties described as follows.

Algorithm 1: Child context algorithm, *child*($\mathcal{E}\ell$)

```
    input  : A schema tree element, $\mathcal{E}\ell$
    output: child context set, $C\_set$
 1  $C\_set \leftarrow \phi$;
 2  if $atomic\_Element\ (\mathcal{E}\ell)$ then
 3      return $C\_set$;
 4  else
 5      $k \leftarrow$ postorder($\mathcal{E}\ell$);
 6      $i \leftarrow$ getIndex($\mathcal{E}\ell$);
 7      repeat
 8          if $k = CPS.NPS[i]$ then
 9              $C\_set \leftarrow C\_set \bigcup CPS.LPS.OID[i]$;
10          end
11          $i \leftarrow i + 1$;
12      until $k < CPS.NPS[i]$ ;
13  end
14  return $C\_set$;
```

- *The child context;* Using the edge relationship property, we can identify immediate children of a complex node and their number. The number of immediate children of a complex node from the *NPS* sequence is obtained by counting its postorder traversal number in the sequence, and then we can identify these children.

  To identify the child context of an element, we first use the atomic element function, as shown in Algorithm 1. If this function returns true, then the element is an atomic (leaf) element and its child context set is empty, *line* 3. If the function returns false, the edge relationship property is used to extract the element child set, *lines* 7 *to* 12. To do this, we first determine the postorder traversal number and the first occurrence of this number in the CPS representations, *lines* 5&6. Then, we scan the CPS representation starting from that index from left to right (conceptually) till the first appearance of a CPS entry whose NPS is greater than the traversal number of the elements. During this scan, we determine the entries whose NPSs equal the postorder traversal number of the element, *line* 8. The elements in the corresponding LPSs represent the child context of the element, *line* 9, and are added to the child context set, *C_set*, of the element. The scanning process will be terminated when there is a CPS entry with NPS greater than the element traversal number.

For example, consider the schema tree $ST1$ of *Example 5.1*, the postorder number of node n_1 is 11. This number occurs three times. This means that it has three immediate children $\{n_2, n_3, n_4\}$. While, the postorder number 6 does not appear in $NPS(ST1)$, this means that the node n_9 is an atomic node (atomic node property) and its child context set is empty.

Algorithm 1 shows several interesting points. The most significant is the limitation of the scanning process. This process does not start from the beginning of the CPS representation, however, it starts at the first appearance of the traversal number of the element and it stops when the index goes one level higher than the element level.

- *The leaf context;* The CPS properties assert that the postorder numbers of atomic elements (nodes) do not appear in the NPS sequence. Considering this assertion and taking into account the child context, we could recursively obtain the leaf context of a given element.

  The leaf context set (*L_set*) can be identified by a procedure similar to that of child context. As shown in *Algorithm* 2, it starts by examining if the element is an atomic element or not using the atomic element property, *line* 1. When the element is a complex element, the algorithm starts by extracting the element child context set, *line* 4. Then, the algorithm iterates through the elements of the child context set, *lines* 6 *to*14. If the element of the child set, $\mathcal{E}\ell_i$, is an atomic element, it is added to the leaf context set, *line* 9, otherwise the function recursively call itself, *line* 11.

  For example, consider the schema tree $ST1$ of *Example 5.1*, nodes $\{n_2, n_3, n_7, n_8, n_9, n_{10}, n_{11}\}$ are the leaf node set. Node n_6 has two children n_7, n_8, which are leaves. They form the leaf context set of node n_6.

- *The sibling context;* Using the sibling relationship property facilitates the identifi-

Algorithm 2: Leaf context algorithm, $leaf(\mathcal{E}\ell)$

```
input  : A schema tree element, $\mathcal{E}\ell$
output: leaf context set, $L\_set$
if $atomic\_Element\ (\mathcal{E}\ell)$ then
    return $\phi$ ;
else
    $C\_set \leftarrow child\ (\mathcal{E}\ell)$;
    $i \leftarrow 1$;
    while $C\_set$ has more elements do
        $\mathcal{E}\ell_i \leftarrow C\_set[i]$;
        if $atomic\_Element\ (\mathcal{E}\ell_i)$ then
            $L\_set \leftarrow L\_set \bigcup \mathcal{E}\ell_i$;
        else
            $leaf\_Context\ (\mathcal{E}\ell_i)$;
        end
        $i \leftarrow i + 1$;
    end
end
return $L\_set$;
```

cation of the sibling context for an element. To this, different procedures can be used. One of these is to first identify the parent of this element using the edge relationship property, and then applying the child context procedure of that parent. For example, consider the schema tree $ST1$ of *Example 5.1*, the node $\{n_9\}$ has a parent element $\{n_5\}$. The child context set of $\{n_5\}$ is $\{n_5, n_9, n_{10}\}$. Therefore, the sibling context set of the element n_9 is $\{n_5, n_{10}\}$.

- *The ancestor context;* To determine the ancestor context of an element, we utilize the CPS properties, as shown in *Algorithm* 3. We first examine if the element is a root, *line*1, atomic, *line*3, or complex, *line*7, element. When the element is a root element, the algorithm returns a *Null* path, *line*2. Otherwise, the algorithm calls a function, *getAncestor*, *lines*4&8, that determines the ancestor context for the element. When the element is an atomic element, the algorithm passes the parent of the specified element instead of it. The *getAncestor* function starts by a set of initialization to get the element postorder number, first occurrence in NPS, *line* 12. For a passed element, we obtain the ancestor context by scanning the *NPS* sequence from the first occurrence of the element in the sequence to the first reach to the root element, *line*19, (conceptually from the left to the right) and by identifying the numbers which are larger than the postorder number of the node until the first occurrence of the root node. While scanning from the left to the right, we ignore nodes whose postorder numbers are less than postorder numbers of already scanned nodes. For an atomic (leaf) node, the ancestor context is the ancestor context of its parent union of the parent node itself. For example, consider the schema tree $ST1$ of *Example 5.1*, the ancestor context of node n_5 (non-atomic node) is the path $n_1/$ $n_4/$ n_5, while the ancestor context of node n_9 (atomic node) is the path n_1/n$_4$/ n$_5$/ n$_9$.

Algorithm 3: Ancestor context algorithm, *ancestor*($\mathcal{E}\ell$)

```
    input  : A schema tree element, $\mathcal{E}\ell$
    output: Ancestor context path, $P_{\mathcal{E}\ell}$
1   if $root\_Element\ (\mathcal{E}\ell)$ then
2       return $Null$;
3   else if $atomic\_Element\ (\mathcal{E}\ell)$ then
4       $P_{\mathcal{E}\ell} \leftarrow \mathcal{E}\ell \cup$ getAncestor (parent($\mathcal{E}\ell$));
5       return $P_{\mathcal{E}\ell}$;
6
7   else
8       $P_{\mathcal{E}\ell} \leftarrow$ getAncestor ($\mathcal{E}\ell$);
9       return $P_{\mathcal{E}\ell}$;
10  end
11  getAncestor ($\mathcal{E}\ell$);
12  $k \leftarrow$ postorder($\mathcal{E}\ell$); $i \leftarrow$ getIndex($\mathcal{E}\ell$);
13  $P_{\mathcal{E}\ell} \leftarrow \mathcal{E}\ell$; $max \leftarrow k$;
14  repeat
15      $k' \leftarrow CPS.NPS[i+1]$;
16      if $k' > max$ then
17          $max \leftarrow k'$; $P_{\mathcal{E}\ell} \leftarrow P_{\mathcal{E}\ell} \bigcup node\,with\,k'$;
18      end
19  until $the\,first\,reach\,to\,the\,root$ ;
20  return $P_{\mathcal{E}\ell}$;
```

This algorithm also presents several interesting points. The most significant one is the limitation of the scanning process. This process does not start from the beginning of the CPS representation, however, it starts at the first appearance of the traversal number of the element and it stops when the index reaches the first occurrence of the root element.

5.5 Summary

As shared XML data proliferates, the need to develop high performance techniques to identify and discover semantic correspondences across these data is always in great demand. The most dominant step is to define, adopt, and utilize similarity measures between XML data elements. In this chapter, we introduced several element similarity measures, and classified them based on the exploited information: the internal element similarity measures that exploit element features, and external measures that consider element surroundings. In order to effectively and efficiently utilize these measures, we proposed the sequence representation of schema trees using the Prüfer encoding method. Finally, we constructed a link between the sequence representation and element similarity measures using properties of CPSs.

6 Sequence-based XML schema matching

In this chapter, we present our matching techniques that are based on the sequence representation of XML schema trees. The key idea is to exploit both semantic and structural information of schema trees in proficient means. Exploiting the semantic and structure information aims at improving the matching quality and effectiveness, while exploiting them in competent means endeavors to improve the matching efficiency. Indeed, the sequence representation of XML schema trees assists to achieve these objectives.

Regarding the matching quality point of view, a matching system has high matching result if the system is able to identify and discover complex matches as well as simple matches. During the matching system design we take this aspect into account. Therefore, we start by developing a matching system able to discover simple matches, then we further widen the system to discover complex matches. We observe that the system becomes more efficient than before during the system extension.

The material presented in this chapter has been developed and published in [17, 15, 14].

6.1 Introduction & motivations

As we mentioned before, the proliferation of shared XML data necessitates the need for high performance techniques to identify and discover semantic correspondences across the XML data. Two equally important aspects should be considered: *matching effectiveness* and *matching efficiency*. Existing schema matching systems such as Cupid [96], COMA/COMA++ [47, 48], LSD/GLUE [50, 49], and S-Match [70] heavily rely on either rule-based approaches or learner-based approaches. In rule-based systems, schemas to be matched are represented as schema trees or schema graphs which in turn requires traversing these trees (or graphs) many times. On the other hand, learning-based systems need much pre-effort to train their learners. As a consequence, especially in large number and large-scale schemas and dynamic environments, matching performance declines radically. Additionally, these systems produce score schema elements, which results in discovering only simple (one-to-one) matches. Discovering one-to-one matches only solves the schema matching problem partially. In order to completely solve the problem, the matching system should discover complex matchings as well as simple ones. Few work has addressed the problem of discovering complex matching [44, 77, 81], because of the greater complexity of finding complex matches than of discovering simple ones. *Therefore, discovering complex matching taking into account schema matching scalability against both the large number of schemas and large-scale schemas is considered a real challenge.*

Motivated by the above challenges and by the fact that the most prominent feature for an XML schema is its hierarchical structure, we propose a novel approach for matching XML schemas. In particular, we develop and implement the *XPrüM* system, which mainly consists of two parts—schema preparation and schema matching. Schemas are first parsed and represented internally using rooted ordered labeled trees, called schema trees. Then, we construct a Prüfer sequence for each schema tree. Prüfer sequences construct a one-to-one correspondence between schema trees and sequences [115]. We capture schema tree semantic information in the *Label Prüfer Sequence (LPS)* and schema tree structural information in the *Number Prüfer Sequence (NPS)*. LPS is exploited by a linguistic matcher to compute terminological similarities among the whole schema elements including both atomic and complex elements. Then, we apply our new structural algorithm schema elements exploiting NPS and previously computed linguistic similarity. The two similarity values are then combined and the top-k mappings are selected to produce the matching result.

To improve the matching result, especially to discover complex matches, we widen the *XPrüM* system. In particular, we introduce the concept of *compatible nodes*.

6.2 The XPrüM system

In this section, we describe the core parts of the XPrüM system. As shown in Fig. 6.1, the system has two main parts: *schema preparation* and *schema matching*. First, schemas are

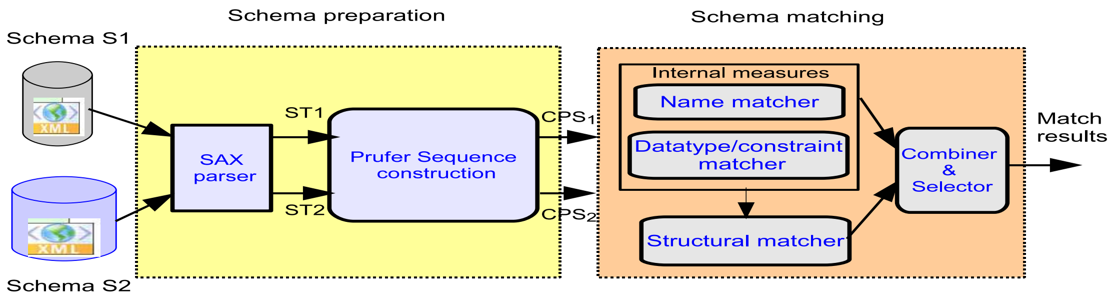


Figure 6.1: Matching process phases

parsed using a SAX parser [1] and represented internally as schema trees. Then, using the Prüfer encoding method, we extract both label sequences and number sequences. Then, the schema matching part discovers the set of matches between two schema elements utilizing both sequences. In the following, we present the core parts of the proposed system in details. The methodologies are discussed via our running example described in Ch.5 (Example 5.1).

6.2.1 Schema preparation

This thesis considers only XML schema matching. However, our approach is a generic framework, i.e., it has the ability to identify semantic correspondences between different models from different domains. In order to make the matching process a more generic process, schemas should be represented internally by a common representation. This uniform representation reduces the complexity of subsequent algorithms by not having to cope with different schema representations. We use rooted ordered labeled trees as the internal model. We call the output of this step the *schema tree (ST)*. As shown in Fig. 6.1, two XML schemas are parsed using the SAX parser and are represented internally as schema trees, *ST1* and *ST2*.

Unlike existing rule-based matching systems that rely on schema trees (or schema graphs) to apply their matching algorithms, we extend schema tree representations into sequences using the Prüfer sequence method, as described in Chapter 5. As a result, each schema tree is represented as a $CPS(NPS, LPS)$, such that the Label Prüfer Sequence, (LPS), captures the semantic information of schema tree elements, while the Number Prüfer Sequence, (NPS), captures the structure information of the schema tree, as shown in Table 5.2.

[1] `http://www.saxproject.org`

6.2.2 Matching algorithms

The proposed matching algorithms operate on the sequential representation of schema trees to discover semantic correspondences between them. Generally speaking, the process of schema matching is performed, as shown in Fig. 6.1, in two phases—*element matchers* (both internal and structural measures) and *combiner & selector*.

Recent empirical analysis shows that there is no single dominant element matcher that performs best, regardless of the data model and application domain [55]. This is due to the fact that schema matching is an intricate problem due to the reasons described in details in Chapter 2. As a result, we should exploit different kinds of matchers. In our approach, we utilize two schema-based matchers—the *internal (linguistic) matcher* and the *external (structural) matcher*, which are based on the element similarity measures described in Chapter 5.

First, a degree of linguistic similarity is automatically computed for all element pairs using the linguistic matcher phase. In our approach, we make use of the *name similarity measure* to exploit element names; the *data type similarity measure* to exploit element types/data types, and the *constraint measure* to exploit element cardinality constraints. Once the degree of linguistic similarity is computed, the second matcher, structural similarity measure, starts to structure similarity between schema elements. After a degree of similarity is computed, in the second phase it is addressed how to combine different similarities from different element matchers and select top$-K$ mappings.

In the following section, we are going to describe matcher algorithms. Without loss of generality, let the number of nodes in *ST1* and *ST2* be n and m, respectively.

6.2.3 Internal matching algorithms

The aim of this phase is to obtain an initial similarity value between the elements of the two schema trees based on the similarity of their features (labels). To this end, we make use of the internal similarity measures described in Chapter 5. In particular, we utilize basic schema-based matchers—the *name similarity measure*, *type/data type similarity measure* and *constraint similarity measure*.

Name similarity algorithm

The name similarity between two (or more) schema tree elements is computed using the name similarity measure described in Chapter 5. Based on this measure, we develop a name similarity algorithm, as shown in *Algorithm* 4. The algorithm accepts two label Prüfer sequences of the two schema trees $ST1$ and $ST2$, $LPS_{ST1}\&LPS_{ST2}$ as input and calculates and constructs an $n \times m$ name similarity matrix, $NsimM$. For each element name of the first label sequence, *line* 3, the algorithm extracts element names from the second sequence, *line* 5, and then it applies the name similarity measure to every element pair, *lines* 6&7&8. Finally, the algorithm uses a weighted sum function to combine the three similarity values, *line* 9.

Algorithm 4: Name similarity algorithm

```
input  : LPS_{ST1} & LPS_{ST2}
output: Name similarity matrix NsimM
1  $NsimM$ [ ][ ]$\leftarrow$ 0;
2  for $i \leftarrow 1$ to $n$ do
3      $s_1 \leftarrow LPS_{ST1}[i].name$;
4      for $j \leftarrow 1$ to $m$ do
5          $s_2 \leftarrow LPS_{ST2}[j].name$;
6          $sim_1 \leftarrow$ sim$_{edit}$($s_1, s_2$);
7          $sim_2 \leftarrow$ sim$_{tri}$($s_1, s_2$);
8          $sim_3 \leftarrow$ sim$_{Jaro}$($s_1, s_2$);
9          $NsimM$ [$i$][$j$] $\leftarrow$ combine$_{name}$($sim_1, sim_2, sim_3$);
10     end
11 end
```

Example 6.1 *Applying Algorithm 4 on CPSs illustrated in Table 5.2, we get a 11×11 NsimM matrix. Of course, these initial candidates contain many false positive matches. For example, ST1.staff correspondences initially with ST2.staff with a name similarity value that equals 1.0.*

Data type & constraint similarity algorithms

The name similarity measure is a necessary measure to produce initial similarity values between schema elements, however, it is insufficient measure, as shown in the above example. To enhance the matching result and to prune some of the false positive candidates, we make use of both data type and cardinality constraint measures, described in Chapter 5. Based on these measures, we propose a data type/constraint algorithm, as shown in *Algorithm* 5.

The algorithm accepts label Prüfer sequences of schema trees, $LPS_{ST1}\&LPS_{ST2}$, and the name similarity matrix computed previously as input and constructs an $n \times m$ linguistic similarity matrix, $LsimM$, as output. For each element in the first sequence, extracting its type and constraint, *lines* 2&3, it extracts types and constraints for all elements in the second sequence, *lines* 5&6,. The algorithm then determines the type similarity (cardinality similarity) between all element type (constraint) pairs using the built type (constraint) similarity table described in Chapter 5, *line* 7(8). We store the type (constraint) similarity values in a type (constraint) similarity matrix, $TsimM(CsimM)$. In fact, we need these similarity values for studying the effect of individual element matchers, as will be shown in the next chapter. Hence, we keep them in a matrix rather than in a single variable. Finally, the algorithm combines the name, type, and constraint similarity values using an aggregation function, *line* 9, and constructs the linguistic similarity matrix, $LsimM$.

Algorithm 5: Datatype/constrain similarity algorithm

```
input  : LPS_{ST1} & LPS_{ST2} & NsimM
output: Linguistic similarity matrix LsimM
1  for $i \leftarrow 1$ to $n$ do
2      $dt_1 \leftarrow LPS_{ST1}[i].datatype$;
3      $car_1 \leftarrow LPS_{ST1}[i].cardinality$;
4      for $j \leftarrow 1$ to $m$ do
5          $dt_2 \leftarrow LPS_{ST2}[j].datatype$;
6          $car_2 \leftarrow LPS_{ST2}[j].cardinality$;
7          $TsimM[i][j] \leftarrow$ Tsim($dt_1, dt_2$);
8          $CsimM[i][j] \leftarrow$ Csim($car_1, car_2$);
9          $LsimM[i][j] \leftarrow Combine_I(NsimM[i][j], TsimM[i][j], CsimM[i][j])$;
10     end
11 end
```

6.2.4 Structural matching algorithms

The matching algorithms described above consider only the label information and ignore the structural information. There can be multiple match candidates which differ in structure but have the same label. The structural matching algorithm prunes these false positive candidates by considering the structural information represented in the *CPS* sequence.

As mentioned in Chapter 5, our structural matcher is motivated by the fact that the most prominent feature in an XML schema is its hierarchical structure, and this matcher is based on the element context described there. The structural node context defined above relies on the notion of path and set. In order to compare two ancestor contexts, we essentially compare their corresponding paths. On the other hand, in order to compare two child contexts and/or leaf contexts, we need to compare their corresponding sets. In the following, we describe how to compute structural context measures:

Algorithm 6: Child context similarity measure, $ChSim(\mathcal{E}\ell_1, \mathcal{E}\ell_2)$

```
   input  : two elements, $\mathcal{E}\ell_1$ & $\mathcal{E}\ell_2$
   output: a child context similarity child_sim
1  if $\mathcal{E}\ell_1$ or $\mathcal{E}\ell_2$ is atomic then
2      return 0;
3  else
4      $sum \leftarrow 0$;
5      $child\_set1 \leftarrow$ child($\mathcal{E}\ell_1$);
6      $child\_set2 \leftarrow$ child($\mathcal{E}\ell_2$);
7      foreach $\mathcal{E}\ell_i$ in child_set1 do
8          $max \leftarrow 0$;
9          foreach $\mathcal{E}\ell_j$ in child_set2 do
10             sim $\leftarrow$ InterSim($\mathcal{E}\ell_i$, $\mathcal{E}\ell_j$);
11             if $sim > max$ then
12                 $max \leftarrow sim$;
13             end
14         end
15         $sum \leftarrow sum + max$;
16     end
17     return average(sum)
18 end
```

Child context similarity algorithm

To obtain the child context similarity between two nodes, we compare the two child context sets for the two nodes. The steps required to achieve this task is illustrated in *Algorithm 6*. The algorithm starts by examining either one of the two elements or both is an atomic element, *line* 1, it returns with child similarity value of 0. If not, the algorithm first extracts the child context set for each node from the *NPS* and *LPS* sequences, *lines* 5&6. Second, we get the linguistic similarity between each pair of children in the two sets, *line* 10. Third, we select the matching pairs with maximum similarity values, *lines* 11 *to* 13. And finally, we take the average of best similarity values, *line* 17.

Leaf context similarity algorithm

To compute the leaf context similarity between two elements, we compare their leaf context sets. To this end, first, we extract the leaf context set for each node. Second, we determine the gap between each node and its leaf context set. We call this vector the gap vector. Third, we apply the cosine measure between two vectors (see *Algorithm 7)*.

Algorithm 7: Leaf context algorithm, $LeafSim(\mathcal{E}\ell_1, \mathcal{E}\ell_2)$

```
input  : two elements, $\mathcal{E}\ell_1$ & $\mathcal{E}\ell_2$
output: a leaf context similarity leaf_sim
1 if $\mathcal{E}\ell_1$ or $\mathcal{E}\ell_2$ is atomic then
2     return 0;
3 else
4     $v_1 \leftarrow$nodegapvector($\mathcal{E}\ell_1$);
5     $v_2 \leftarrow$nodegapvector($\mathcal{E}\ell_2$);
6     return $CM(v_1, v_2)$
7 end
```

Sibling context algorithm

To get the sibling context similarity, we use an algorithm similar to the algorithm, *Algorithm 6*, used to compute the child context similarity, except that this algorithm extracts the sibling context sets instead of the child context sets.

Ancestor context algorithm

Algorithm is used to assess the ancestor similarity between every element pair. If one or both of them is a root element, *line* 1 , the algorithm returns a value of 0, *line* 2. Otherwise, it extracts the ancestor path for each element, *lines* 3&4. Then the algorithm uses the equation (5.14) to compute the path similarity between two ancestor pathes, *line* 6.

Putting it all together

Our complete structural matching algorithm is shown in *Algorithm 9*. The algorithm accepts *CPS(NPS, LPS)* for each schema tree and the linguistic similarity matrix as inputs to produce a structural similarity matrix, $SsimM$. For all element pairs (*lines* 2&3) the

Algorithm 8: Ancestor context algorithm, $PSim(\mathcal{E}\ell_1, \mathcal{E}\ell_2)$

```
input  : two elements, $\mathcal{E}\ell_1$ & $\mathcal{E}\ell_2$
output: ancestor context similarity p_sim
if $\mathcal{E}\ell_1$ or $\mathcal{E}\ell_2$ is a root then
    return 0;
else
    $p_1 \leftarrow$ancestor($\mathcal{E}\ell_1$);
    $p_2 \leftarrow$ancestor($\mathcal{E}\ell_2$);
    return $PSim(v_1, v_2)$
end
```

context similarity is computed using child context similarity algorithm(Algorithm 6), ancestor context similarity algorithm (*line* 5) (Algorithm 8) leaf context similarity algorithm (Algorithm 7), and the sibling context similarity algorithm. The context similarity values are then combined (*line* 7) using an aggregation function.

Algorithm 9: Structural similarity algorithm

```
input  : CPS$_{ST1}$(NPS, LPS) & CPS$_{ST2}$(NPS, LPS) & $LsimM$
output: Structural similarity matrix $SsimM$
$SsimM[][] \leftarrow 0$;
for $i \leftarrow 1$ to $n$ do
    for $j \leftarrow 1$ to $m$ do
        $sim_1 \leftarrow$ChSim($\mathcal{E}\ell_i$, $\mathcal{E}\ell_j$);
        $sim_3 \leftarrow$SibSim($\mathcal{E}\ell_i$, $\mathcal{E}\ell_j$);
        $sim_2 \leftarrow$LeafSim($\mathcal{E}\ell_i$, $\mathcal{E}\ell_j$);
        $sim_4 \leftarrow$PathSim($\mathcal{E}\ell_i$, $\mathcal{E}\ell_j$);
        $SsimM[i][j] \leftarrow$Combine$_E$($sim_1, sim_2, sim_3, sim_4$);
    end
end
```

6.2.5 Combining element similarity measures

As shown in the previous sections both the internal and external measures encompass multiple measures. This multi-measure nature is potent in that it makes a matching system highly flexible and adaptable to a particular application domain. However, it results in considerable challenges on how to combine these measures. Without a proper means of combining, the element similarity measures fail to produce correct correspondences across

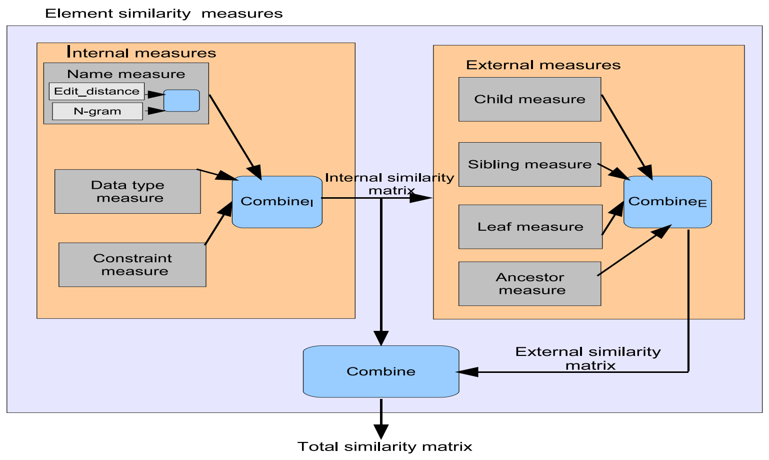


Figure 6.2: Similarity combining scenarios.

schema elements. To get a total similarity value between pairs of schema elements, we should combine their individual similarity values resulted from the internal and external measures. Indeed, combining similarity values are needed in the following scenarios, as shown in Fig. 6.2:

1. Within the individual measures, the name measure consists of three measures (Levenshtein distance, Jaro, and tri-gram) for determining the degree of syntactic similarity. To get the name similarity value, these individual measures should be then combined;

2. Within either the internal measure or the external measure, the internal measure comprises three element similarity measures (name, data type, and cardinality). Consequently, to obtain a total similarity value of the internal measure, the similarity values produced by these element measures should be combined. For example, the function $Combine_I$ in Definition 5.5 is a combination function used to combine similarity measures of the internal measure, as shown in Fig. 6.2, and

3. Within the element measure; An element measure composes of two parts, internal and external. As a result, a total similarity value between pairs of schema elements is obtained by combining similarity values produced by both parts.

Similarity combining is far from being trivial, due to the wide variety of combining techniques employed. Furthermore, the number of element similarity measures is continuously growing, and this diversity by itself complicates the choice of the most appropriate combining strategy for a given application domain. Commonly, similarity combining can be done using aggregation functions that can be defined as follows [26]:

Definition 6.1 *An aggregation function $\mathcal{F}$ is a function of $n > 1$ arguments that maps the (n-dimensional) unit cube onto the unit interval $\mathcal{F} : [0,1]^n \rightarrow [0,1]$, with the following properties*

1. $\mathcal{F}(\underbrace{0, 0, \cdots, 0}_{n-times}) = 0$ *and* $\mathcal{F}(\underbrace{1, 1, \cdots, 1}_{n-times}) = 1$.
2. $x \preceq y$ *implies* $\mathcal{F}(x) \leq \mathcal{F}(y)$ *for all* $x, y \in [0,1]^n$

The input value of 0 is interpreted as no membership, or no evidence, and naturally, an aggregation of $n\, 0s$ yields 0. Similarly, the value of 1 is interpreted as full membership, and an aggregation of $n\, 1s$ yields 1. This is the first fundamental property of aggregation functions, the preservation of bounds. The second property is the monotonicity condition. The condition can be stated that for every $x_1, x_2, \cdots, x_n \in [0,1]$ and $y_1, y_2, \cdots, y_n \in [0,1]$ such that $x_i \leq y_i$, $\mathcal{F}(x_1, x_2, \cdots, x_n) \leq \mathcal{F}(y_1, y_2, \cdots, y_n)$.

In the literature, many aggregation functions have been proposed. The question is how to choose the most suitable aggregation function for a specific application. In the context of schema matching, the existing aggregation methods focus on linear combination functions, which cannot sufficiently explore the interdependencies between different element similarities [48, 55, 97]. Moreover, a single (fixed) aggregation function is always applied to combine similarities from different schemas without considering the special features of each schema pair. To the best of our knowledge, no work has proposed the use of nonlinear combination to aggregate the element similarity measures. To this context, we propose and evaluate different combination strategies to combine element measures.

- *Linear fixed methods*; These methods only consider the effect of individual similarities without considering the interdependencies between them. One of the following methods can be used [47, 48]:
  - **Max**. This method selects the largest similarity value of any measure.
  - **Min**. This method selects the smallest similarity value of any measure.
  - **Weighted-sum**. This method determines a weighted sum of similarity values of the individual measures and needs relative weights which should correspond to the expected importance of the measures. The weighted-sum function can be defined as follows

Definition 6.2 *Given a weighted vector w, such that $\sum_{i=1}^{n} w_i = 1$, the weighted-sum function is the function $\mathcal{F}_w(X) = w_1x_1 + w_2x_2 \cdots w_nx_n = \sum_{i=1}^{n} w_ix_i$*

– **Average**. This method represents a special case of Weighted-sum and returns the average similarity over all measures, i.e. considers them equally important.

As seen, these methods focus on linear combination functions, which cannot sufficiently explore the interdependencies between element measures. To address this problem, we propose using the nonlinear methods. Furthermore, the linear fixed methods make use of constant weight values, which need to be specified manually in most cases. To make weight values adaptable, the linear adaptive methods can be used.

- *Linear adaptive methods*; These methods do not also consider the interdependencies between similarity measures, however, they deem the adaptive behavior of weights. The values of weights should be determined through a learning process using a learning technique. The similarity combining function, $\mathcal{F}_w(X) = \sum_{i=1}^{n} w_ix_i$, is considered as an optimization problem, where the weights should be chosen to maximize $\mathcal{F}_w(X)$. The Hopfield Neural Network [78] or the machine-learning techniques [97] can solve these optimization problems.

- *Nonlinear methods*; These methods consider the interdependencies between element measures by using nonlinear techniques, such as nonlinear networks. Since the similarity values are ranging between 0 and 1, so the similarity combining function should be restricted to the second order. In this direction, the similarity combining function can be defined as follows:

$$Sim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right) = \lambda \sum_{i=1}^{\mathcal{N}} w_i Sim_i\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right)$$
$$\pm(1-\lambda) \sum_{j=1}^{\mathcal{N}-1} \sum_{k=j}^{\mathcal{N}} Sim_j\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right) Sim_k\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right)$$

The first term in this equation presents similarity combining without considering the similarity measures interdependencies, while the second part presents these interdependencies. The equation also shows that the two values are either added or subtracted depending on the linear similarity value. The intuition behind this is that the higher (linear) similarity between elements is, elements are likely to be similar and the two parts should be added. In contrast, the low similarity, elements are not likely to similar and the two parts should be subtracted. Finally, the constant λ is used to ensure the total similarity value in the normalized range, i.e., [0,1].

In the thesis, we make use of the first and the third methods and experimentally compare between them, while the second method is left for future work.

After obtaining the match similarity matrix, an element selector is used to select matches from the given matrix. In fact, several methods can be used. The simplest selection strategy is *thresholding*: all pairs of elements with a similarity value exceeding a given threshold are returned as matches [47, 48]. More complex strategies include formulating the selection as an optimization problem over a weighted bipartite graph [55]. In our implementation we make use of the thresholding strategy.

6.2.6 Complexity analysis

The worst case time complexity of the *XPrüM* system can be expressed as a function of the number of nodes and the number of input schemas. Let n be the average schema size and S be the number of input schemas. Following the same process in [62], we can prove the overall time complexity of our system as follows:

- *Prüfer sequences construction*; Schemas to be matched are first postorder traversed and represented as CPSs with a time complexity of $O(nS)$.
- *Internal matching algorithms*; This phase requires a comparison between the whole schema nodes with a time complexity of $O(n^2S^2)$.
- *Structure matching algorithms*; This phase also requires a comparison between the whole schema nodes with a time complexity of $O(n^2S^2)$.

Along the light of these observations, the overall worst-case time complexity of the *XPrüM* system is $O(n^2S^2)$. This is due to the large complexity of linguistic matching. However, the system shows additional improvements specially in the other phases. The following chapter experimentally confirms this complexity.

6.3 EXPrüM: Enhancement of XPrüM

Although simple (one-to-one) matching has got great attention, complex matching has not been extensively investigated, mainly due to the much more complex search space of exploring all possible combinations of schema elements. Furthermore, simple matches are common, however, they only solve the schema matching problem partially. In order to completely solve the problem, a matching system should discover complex matches as well as simple ones.

for the following reason [44], discovering complex matches is fundamentally harder than simple matches. While the number of candidate 1-1 matches between a pair of schemas is bounded (by the product of the sizes of the two schemas), the number of candidate complex matches is not. There is an unbounded number of functions for combining elements in a schema, and each one of these could be a candidate match. Hence, in addition to the inherent difficulties in generating a match to start with, the problem is exacerbated by having to examine an unbounded number of match candidates. Consider two schema trees with n and m elements, respectively, while there $n \times m$ simple matches, the number of possible complex matches is exponential. Therefore, many systems including *XPrüM* have been developed focusing on the problem of discovering simple matching, while, a few matching systems have addressed the problem of discovering complex matches in small-scale schemas [44, 77, 145, 81]. *Discovering complex matching taking into account schema matching scalability against both a large number of schemas and large-scale schemas is considered a real challenge.*

In this thesis, we assume that simple matches exist and should be discovered among either simple and/or complex elements, while complex matches are discovered only among simple elements. The motivation behind this assumption is that a complex element comprises simple and/or complex elements. We notice that it rarely occurs that a complex element in a schema corresponds to more than one complex element in another schema. However, it is obvious for one (or more) simple element in one schema to correspond to one or more simple elements from another schema.

To this end, we modify the *XPrüM* system to be able to discover complex matches considering large-scale schemas. As shown in Fig. 6.3, the *EXPrüM* system has two main parts compared to *XPrüM*: schema preparation and schema matching. The schema preparation part, as described before, accepts XML schemas and presents them as sequences, $CPSs$, using the Prüfer encoding method, while the schema matching part has a slightly different structure than that in the original system. It contains three phases—*linguistic*

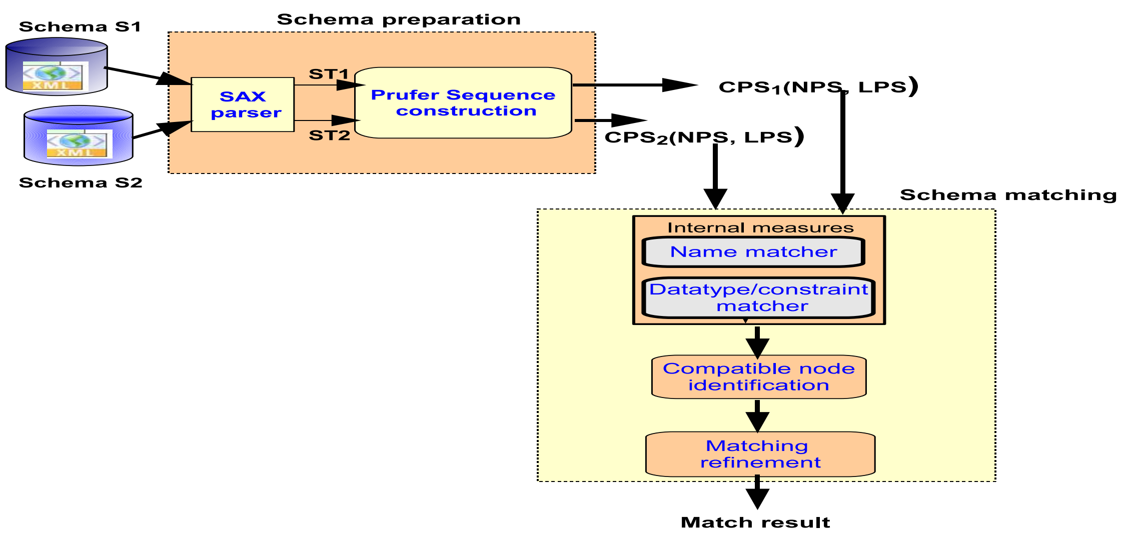


Figure 6.3: EXPrüM system

matching, *compatible elements identification*, and *matching refinement*.

First, a degree of a linguistic similarity matrix, $LSimM$, is automatically computed for all element pairs using the internal (linguistic) matcher phase, as in the XPrüM system. Once the degree of linguistic similarity is computed, the second phase starts to identify compatible elements (nodes). In this phase, we apply our structural matcher only to complex elements. Then we combine both linguistic and structural similarity measures for complex elements and select compatible elements among them. Finally, a matching refinement phase is carried out to discover atomic elements inside each compatible node pair. The set of compatible elements and semantic corresponding atomic elements constitutes the matching result.

6.3.1 Compatible elements identification

Before detailing the process to identify and discover compatible elements (nodes), we define what compatible elements are.

Definition 6.3 *Let $\mathcal{E}\ell_i \in ST1$ and $\mathcal{E}\ell_j \in ST2$ be two complex elements. If the computed similarity (linguistic and structural) exceeds a predefined threshold th, $Sim(\mathcal{E}\ell_i, \mathcal{E}\ell_j) \geq th$, then, the two elements are compatible elements.*

Unlike state-of-the-art approaches, we first apply our structural algorithm only to complex nodes to compute structural similarities between them, assuming that no match

Table 6.1: Similarity values Top-3 ranking for each node

ST1	*ST2*	*similarity value*	*status*
n_1	n_1	0.624	✓
	n_5	0.4950	
	n_4	0.4837	
n_4	n_3	0.5816	✓
	n_5	0.457	
	n_1	0.4406	
n_5	n_4	0.6145	✓
	n_3	0.500	
	n_1	0.4863	
n_6	n_5	0.524	✓
	n_3	0.514	
	n_4	0.482	

between atomic and complex nodes exists. Then, we combine both linguistic similarity and structural similarity for complex nodes using the weighted sum aggregation. Due to uncertainty inherent in schema matching, the best matching can actually be an unsuccessful choice [66]. To overcome this shortcoming, matching candidates are ranked up to top-3 ranking for each element. Then, we select matching candidates that exceed a threshold, which equals to the smallest similarity value of a true positive candidate. The resulting matching set constitutes a set of compatible element pairs.

Example 6.2 *Considering the two schema trees shown in Fig. 5.1 and their CPSs illustrated in Table 5.2. Table 6.1 represents top-3 ranking, where a check mark ✓ in the status column denotes a true positive match, whereas an empty cell stands for a false positive match. Let there be a threshold value of 0.524, then the set of compatible node pairs is* $\{(ST1.n_1, ST2.n_1), (ST1.n_4, ST2.n_3), (ST1.n_5, ST2.n_4), (ST1.n_6, ST2.n_5)\}$.

6.3.2 Matching refinement

By identifying compatible nodes and the category set for each element, we have obtained top-level matchings (complex elements). In the following, we continue with bottom-level matchings (atomic nodes). We have already computed the linguistic similarity of these nodes, now we have to compute their structural similarity. In this phase, we ought not to carry out the structural similarity algorithm on all simple nodes. Compatible elements (similar complex elements) have the chance to bear similar simple nodes. Along this

Table 6.2: category set for each compatible node

ST1		*ST2*	
Comp. node	Category set	Comp. node	Category set
n_1	C_1={n_2, n_3}	n_1	C_1={n_2}
n_4	C_2={n_{11}}	n_3	C_2={n_{11}}
n_5	C_3={n_9, n_{10}}	n_4	C_3={n_9, n_{10}}
n_6	C_4={n_7, n_8}	n_5	C_4={n_6, n_7, n_8}

light of thinking, we apply structural algorithms on simple nodes inside every compatible element pair. To this, we first give a definition for the compatible element category.

Definition 6.4 *A category of a given compatible element $\mathcal{E}\ell_i$ is a set of elements including*

- *all immediate atomic children nodes of $\mathcal{E}\ell_i$,*
- *all non-compatible (complex) nodes which are immediate children of $\mathcal{E}\ell_i$ and their atomic children.*

Example 6.3 *Considering the two schema trees and their compatible nodes represented in Example 5.1, Table 6.2 illustrates these compatible nodes and the associated category for each one.*

In general, atomic elements neither have a child context nor a leaf context. Therefore, to compute structural similarity for atomic elements, we only compare nodes in each compatible category pair using the ancestor context algorithm presented in the previous section. For example, the category *ST1.*C_1= {n_2, n_3} is only compared to its compatible category *ST2.*C_1= {n_2}. At first, we extract the ancestor context for each node. Consider P_2 represents the ancestor context of *ST1.*n_2 and P'_2 represents the ancestor context of *ST2.*n_2. Then, the structural similarity between the two nodes is given by

$$SSim(ST1.n_2, ST2.n_2) = PSim(P_2, P'_2) \tag{6.1}$$

where $PSim(P_2, P'_2)$ is computed using Equation 5.15. Then, we combine both linguistic and structural similarities using a weighted sum function and select the best candidate(s) based on a predefined threshold.

By this mechanism we gain two main advantages.

- First, we reduce the search space complexity for atomic nodes.
- Second, many false positive candidates are pruned. Furthermore, we can easily discover complex matchings.

Discovering complex matchings

XPrüM identifies element-level matchings between either atomic or complex elements. This solves the schema matching problem partially. To fully solve the problem, we should cope with complex matchings.

Definition 6.5 *If one or more nodes in a category C_i from the source schema correspond with two or more nodes in a compatible category C_j from the target schema, the resulting match is a complex match.*

Example 6.4 *From our running example, the two categories ST1.C_1 and ST2.C_1 are compatible (see Table 6.2). Applying matching algorithms on their nodes, we obtain the complex match, ST2.C_1.n_2 matches (ST1.C_1.n_2, ST1.C_1.n_3). Indeed, the Courses element in the second schema ST2 is the union of the two UnderGrdCourses and GradCourse elements of the first schema ST1. Moreover, the two categories ST1.C_4 and ST2.C_4 are compatible (see Table 4). Applying matching algorithms on their nodes, we obtain the complex match, ST1.C_4.n_7 matches (ST2.C_4.n_6, ST2.C_4.n_7). The name element ($ST1.n_7$) is the concatenation of the two elements FirstName and LastName from the second schema.*

6.3.3 Complexity analysis

As derived in *XPrüM* system, the worst case time complexity of the *EXPrüM* system can also be expressed as a function of the number of nodes and the number of input schemas. Let n be the average schema size and S be the number of input schemas, we can prove the overall time complexity of the enhanced system as follows:

- *Prüfer sequences construction*; As in *XPrüM*, input schemas are first post-order traversed and represented as CPSs with a time complexity of $O(nS)$.
- *Linguistic matching phase*; This phase requires a comparison between all schema elements with a time complexity of $O(n^2S^2)$.

- *Compatible elements identification*; Intuitively, the number of complex nodes is less than the number of atomic nodes in an XML schema tree. Consider this number is given by $c = \frac{n}{\mathcal{N}}$, where $\mathcal{N}$ is an integer number showing the ratio of complex nodes to the total nodes. The compatible nodes identification phase needs to compare only complex nodes with a time complexity of $O(c^2)$ ($\ll O(n^2)$).

- *Matching refinement*; In this phase, we only compare atomic nodes inside a category with atomic nodes inside the corresponding compatible category. Consider the number of compatible nodes c' ($\leq c$) and each category contains n' ($\ll n$) atomic nodes. This phase is performed with a time complexity of $O(c'\ n'^2)$.

Along the light of these observations, the overall worst-case time complexity of the *EXPrüM* system is $O(n^2S^2)$. This is due to the large complexity of linguistic matching. However, the system shows additional improvements specially in the other phases. The following chapter experimentally confirms this complexity.

6.4 Summary

With the emergence of XML as a standard for information representation, analysis, and exchange on the Web, the development of automatic techniques for XML schema matching will be crucial to their success. In this chapter, we have addressed an intricate problem associated to the XML schema matching problem—discovering complex matching considering matching scalability. To tackle this, we have proposed and developed the *XPrüM* system, a hybrid matching algorithm, which automatically discovers semantic correspondences between XML schema elements. The system starts by transforming schemas into schema trees and then constructs a consolidated Prüfer sequence, which constructs a one-to-one correspondence between schema trees and sequences. We capture schema tree semantic information in Label Prüfer Sequences and schema tree structural information in Number Prüfer Sequences. *XPrüM* is defined to identify element level matchings, therefore, we have extended it to discover both simple and complex matches. The enhanced version, *EXPrüM* introduces the concept of compatible elements that provides the process of discovering complex matches. During the development of our systems, we have presented the detailed description of implementation algorithms and analyzed the time complexity of proposed systems. In the following chapter, we present the experimental evaluation that validates the system's performance.

7 Implementation & evaluation

There are a myriad of schema matching approaches and prototypes, and there has been a growing need for evaluating these methods. Matching systems are difficult to compare, but we, well as [124], believe that the schema matching field can only evolve if evaluation criteria are provided and satisfied. These should help system designers to assess the strengths and weaknesses of their systems and help application developers to choose the most appropriate algorithm.

In this chapter, we introduce the experimental evaluation to validate the performance of our proposed schema matching approach. We first present the evaluation measures and criteria used during the evaluation process introducing a new measure, *cost-effectiveness* to combine two performance aspects. We then show several evaluation scenarios considering used data sets and experimental results.

The algorithms described in this thesis have been implemented using Java. We ran all our experiments on a 2.4 GHz Intel core2 processor with 2 GB RAM running Windows XP.

The material presented in this chapter has been developed and published in [17, 5, 15, 9].

7.1 Evaluation measures

Our matching systems *XPrüM* and its enhanced version *EXPrüM* are concerned with both performance aspects—*matching effectiveness (quality)* and *matching efficiency*. Therefore, we have carried out two sets of experiments. The first set demonstrates the ef-

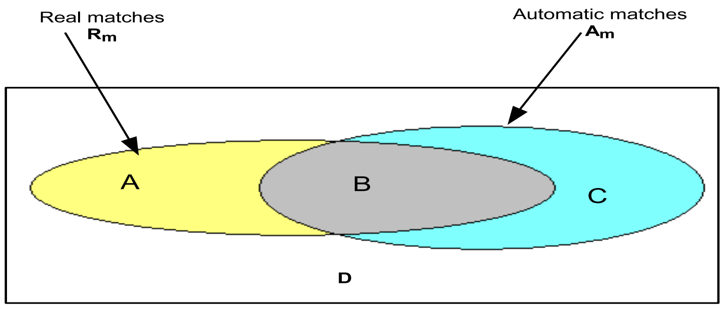


Figure 7.1: Complete set of correspondences.

fectiveness of our matching system, while the second one investigates the system efficiency. To evaluate our matching system, we utilize both performance aspects.

7.1.1 Effectiveness measures

First, the match task should be manually solved to get the real correspondences (matches) R_m. Then, the matching system solves the same problem to obtain automatic correspondences (matches) A_m. After identifying both real and automatic matches, it is possible to define some terms that will be used in computing match effectiveness, as shown in Fig. 7.1. *False negatives* $A = R_m$ - A_m: are the needed matches but not identified by the system; *True positives* $B = R_m \cap A_m$: are the correct matches and identified correctly by the system; *False positives* $C = A_m$ - R_m: are the false matches but identified by the system; and *True negatives* D: are the false matches and correctly discarded by the system. To measure the effectiveness of the matching result, we use the same measures used in the literature, including the following.

- **Precision & Recall.** Precision and recall are the most prominent criteria used to evaluate matching effectiveness. They originate from the information retrieval (IR) field [23, 126] and are based on and can be computed from *real* and *automatic* matches.

  Precision P is meant to determine the degree of correctness of the matching system. It measures the ratio of correctly identified matches (true positives, B) over the total number of identified matches (automatic matches, A_m). It can be computed from

$$P = \frac{|B|}{|B| + |C|} \tag{7.1}$$

Recall R is meant to determine the degree of completeness of the matching system. It measures the ratio of correctly identified matches (true positives, B) over the total number of correct matches (correct matches, R_m). It can be computed from

$$R = \frac{|B|}{|B| + |A|} \tag{7.2}$$

However, neither precision nor recall alone can accurately assess the match quality. Since precision evaluates the post-match effort needed to remove false positives, while recall evaluates the post-match effort needed to add true negatives from the final match result. Hence, it is necessary to consider a trade-off between them. There are several methods to handle such a trade-off, one of them is to combine both measures. The most used combined measures are:

- **F-measure.** *F-Measure* is the weighted harmonic mean of precision and recall. The traditional F-measure or balanced F-score is:

$$F = \frac{2 * |B|}{(|B| + |A|) + (|B| + |C|)} = 2 * \frac{P * R}{P + R} \tag{7.3}$$

  Two other commonly used F-measures are the F_2 measure, which weights recall twice as much as precision, and the $F_{0.5}$ measure, which weights precision twice as much as recall.

- **Overall.** *Overall* is developed specifically in the schema matching context and embodies the idea to quantify the effort needed to add false negatives and removing false positives. It is introduced in the Similarity Flooding (SF) system [101] and is given by

$$OV = 1 - \frac{|A| + |C|}{|A| + |B|} = R * (2 - \frac{1}{P}) \tag{7.4}$$

  For the same precision and recall values, F-measure has higher values than overall. The values of F-measure are always positive and equal zero only when either precision or recall has zero value, while overall has negative values when $P < 0.5$ (i.e., if the number of false positives is larger than the number of true positives). Overall may be zero at different precision values ,e.g. $P = 0.5$. Both measures reach their maximum values of 1.0 when $P = R = 1$. In all other cases, the value of F-measure is within the range determined by precision and recall, while overall is smaller than both precision and recall. Finally, F-measure weights both precision and recall equally, i.e., the effort to add false negatives and the effort to remove false positives, while the overall measure is more sensitive to precision than recall.

7.1.2 Efficiency measures

Efficiency is mainly contained in two properties: *speed* (the time it takes for an operation to complete), and *space* (the memory or non-volatile storage used up by the construct). Speed should be measured in the same conditions, i.e., same processor and same memory consumption for all the systems. If user interaction is required, it has to be ensured that only the processing time of the matching algorithm is measured. In this thesis, we take the response time ($\mathbb{T}$) as an indicator for the schema matching efficiency.

7.1.3 Combining effectiveness and efficiency for schema matching evaluation

Many real-world problems, such as the schema matching problem, involve multiple measures of performance, which should be optimized simultaneously. Optimal performance according to one objective, if such an optimum exists, often implies unacceptably low performance in one or more of the other objective dimensions, creating the need for a compromise to be reached. In the schema matching problem, the performance of a matching system involves multiple aspects, among them effectiveness and efficiency. Optimizing one aspect, for example, effectiveness will affect the other aspects such as efficiency. Hence, we need a compromise between them, and we could consider the trade-off between effectiveness and efficiency matching result as a multi-objective problem. In practice, multi-objective problems have to be reformulated as a single objective problem.

To this end, in this thesis, we propose a method for computing the *cost-effectiveness* of a schema matching system. Such a method is intended to be used in a combined evaluation of schema matching systems. This evaluation concentrates on the *cost-effectiveness* of schema matching approaches, i.e., the trade-off between effectiveness and efficiency. The motivation behind this is that suppose we want to compare two schema matching systems to solve a specific matching problem. Suppose that we have a schema matching problem P, and two matching systems *SA* and *SB*. The system *SA* is more effective than system *SB*, while the system *SB* is more efficient than system *SA*. The arising question here is which system to use to solve the given problem. So far, most existing matching systems [47, 96, 101] only evaluate their performance according to effectiveness issues, hence they all choose the system SA (more effective).

Combining effectiveness and efficiency

From the above criteria we could conclude that the trade-off between effectiveness and efficiency of a schema matching system is considered as a *multi-objective optimization problem (MOOP)*. In this section, we present a definition for the $MOOP$ and the approaches used to solve the problem [151, 98]. In the following definitions we will assume minimization (without loss of generality).

Definition 7.1 *(Multi-objective Optimization Problem) A MOOP is defined as "Find x that minimizes $F(X) = (f_1(x)), f_2(x), ..., f_K(x))^T$ s.t. $x \in S$ and $x = (x_1, x_2, ..., x_n)^T$ where $f_1(x), f_2(x), ..., f_k(x)$ are the k-objective functions, $(x_1, x_2, ..., x_n)$ are the n optimization parameters, and $S \in R^n$ is the solution.*

In our approach, we have two objective functions, overall, OV, (or F-measure) as a measure of effectiveness, and time, $\mathbb{T}$, as a measure of efficiency. Therefore, we could rewrite the multi-objective function as: $CE = (f_1(OV), f_2(\mathbb{T}))$, where CE is the cost-effectiveness which has to be maximized here. In a multi-objective problem, the optimum solution consists of a set of solutions, rather than a single solution as in global optimization. This optimal set is known as the Pareto Optimal set and is defined as follows: $P := \{x \in S| \ \exists x' \in S \ F(x') \preceq F(x)\}$. Pareto optimal solutions are known as the non-dominated or efficient solutions.

There are many methods available to tackle multi-objective optimization problems. Among them, we choose *priori articulation of preference information*. This means that before the actual optimization is conducted, the different objectives are somehow aggregated to one single figure of merit. This can be done in many ways, we choose *weighted-sum approaches*.

Weighted-sum approaches: The easiest and perhaps most widely used method is the weighted-sum approach. The objective function is formulated as a weighted function, as given $min(\, or \, max) \sum_{i=1}^{k} w_i \times f_i(x) \ s.t. \ x \in S$ and $w_i \in R \, |w_i > 0, \sum w_i = 1$. By choosing different weightings for the different objectives, the preference of the application domain is taken into account. As the objective functions are generally of different magnitudes and units, they first should be normalized.

The cost-effectiveness of schema matching

Consider we have two schema matching systems SA and SB to solve the same matching problem. Let OV_A and $\mathbb{T}_A$ represent the overall and time measures of the system SA respectively, while OV_B and $\mathbb{T}_B$ denote the same measures for the system SB.

To analyze the cost-effectiveness of a schema matching system, we make use of the *MOOP* and its method to solve it, namely the *weighted-sum approach*. Here, we have two objectives, namely effectiveness (measured by overall OV) and efficiency (measured by response time $\mathbb{T}$). Obviously, we can not directly add up an overall value to a response time value, since the resulting sum would be meaningless due to the difference of dimensional units. The overall value of a schema matching system is a normalized value, i.e., its range is between 0 and 1 (considering Precision > 0.5), while the processing time is measured in seconds. Therefore, before summing (e.g., weighted average) the two quantities, we should normalize the processing time.

To normalize the response time, for instance, the response time of the slower system (here $\mathbb{T}_A$) is normalized to the value 1, while the response time of the faster system ($\mathbb{T}_B$) can be normalized to a value in the range [0,1] by dividing $\mathbb{T}_B$ and $\mathbb{T}_A$, i.e., $\frac{\mathbb{T}_B}{\mathbb{T}_A}$.

We name the objective function of a schema matching system the *cost-effectiveness (CE)* and it should be maximized. The cost-effectiveness is given by

$$CE = \sum_{i=1}^{2} w_i \times f_i(x) = w_1 \times OV_n + w_2 \times \frac{1}{\mathbb{T}_n} \tag{7.5}$$

where w_1 is the weighting for the overall objective and denoted by (w_{ov}) and w_2 is the weighting for the time objective and denoted by (w_t). In case of comparing two schema matching systems, we have the following normalized quantities OV_{An}, OV_{Bn}, $\mathbb{T}_{An}$ and $\mathbb{T}_{Bn}$ where $OV_{An} = OV_A$, $OV_{Bn} = OV_B$, $\mathbb{T}_{An} = 1$, and $\mathbb{T}_{Bn} = \frac{\mathbb{T}_B}{\mathbb{T}_A}$.

We now endeavor to come up with a single formula involving two quantities, namely *normalized overall* OV_n and *normalized response time* $\mathbb{T}_n$, where each of these quantities is associated with a numerical weight to indicate its importance for the evaluation of the overall performance and to enrich the flexibility of the method. We write the equations that describe the cost-effectiveness (CE)for each system as follows:

$$CE_A = w_{ovA} * OV_{An} + w_{tA} * \frac{1}{\mathbb{T}_{An}} \tag{7.6}$$

$$CE_B = w_{ovB} * OV_{Bn} + w_{tB} * \frac{1}{\mathbb{T}_{Bn}} \tag{7.7}$$

where w_{ov} and w_t are the numerical weights for the overall and time response quantities, respectively. If we let the time weights equal to zero, i.e., w_t=0, then the cost-effectiveness becomes the same normal evaluation considering only the effectiveness aspects (w_{ov}=1).

The most cost-effectiveness schema matching system is the system having a larger CE as measured by the above formulas. Equations 7.6 and 7.7 present a simple but a valuable method to combine the effectiveness and the efficiency of a schema matching system. Moreover, this method is based on and supported by a proven and verified method; the multi-objective optimization problem. Although the method is effective, it still has an inherent problem. It would be difficult to determine good values for the numerical weights, since the relative importance of overall and time response is highly domain-dependent and, of course, very subjective. For example, when we are dealing with small-scale schemas, the overall measure is more dominant than the response time. Hence, we may select w_{ov}=0.8 and w_t=0.2. For the critical time systems, the response time may have the same importance as the overall measure, then we may choose $w_{ov} = w_t$=0.5.

To accommodate this problem, we need an optimization technique which enables us to determine the optimal (or close to optimal) numerical weights. In this thesis, we set these values manually in the selected case studies. Automatic determination of numerical weight values is left for future work.

7.2 Evaluation scenarios

The main objective of this thesis is to develop a matching system that is able to discover simple matches as well as complex ones considering large-scale schemas. Along this light of thinking, we conducted intensive sets of experiments aiming to:

- evaluate the effect of individual element measures on matching performance,
- validate the match effectiveness,
- validate the matching efficiency, and
- compare our system with two well-known matching systems, namely *COMA++* [48, 21, 47] and *Similarity Flooding (SF)* [101].

To realize these goals, we have conducted three main sets of experiments. The first set is to evaluate individual XML element similarity measures, the second set is used to evaluate the *XPrüM* system, while the third set is conducted to validate the performance of the *EXPrüM* system. Both the first and the second sets of experiments have been conducted considering both linear and nonlinear combining strategies, while the third set has been conducted making use of the linear combining strategy.

7.3 First scenario: Element similarity measures evaluation

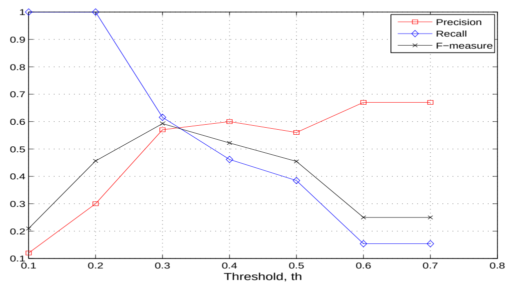
Our strategy in this scenario is to evaluate XML element similarity measures using different settings: (1) every element similarity measure alone and (2) different combinations of similarity measures. Furthermore, the sets of experiments in this scenario have been conducted considering linear and nonlinear combining strategies. We aim to draw general conclusions from this scenario that could be used as a guide through the other scenarios. The quality of the element similarity measures is verified using *precision (P)*, *recall (R)*, and *F-measure*. To realize these goals, we experimented with the two schemas described in Fig. 5.1 and their properties in Table 5.2.

7.3.1 Experimental results

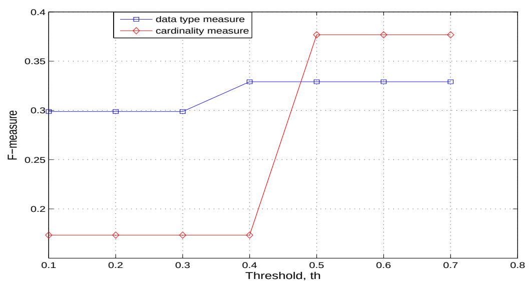
Quality of element similarity measures

1. **Internal measures without external information sources**.

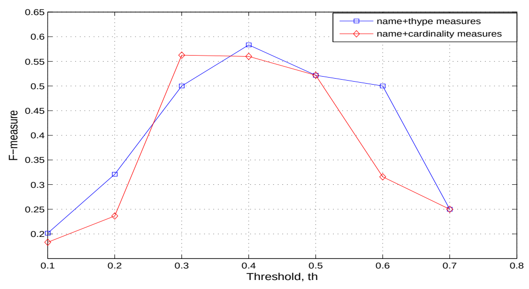
    - Using the linear fixed combination strategy.
      We implemented a set of experiments to observe the quality of internal element similarity measures without exploiting external information sources. The quality of each internal similarity measure (name, data type, and cardinality constraint) is first evaluated alone and then different combinations between them are also evaluated using the linear fixed combination strategy. The similarity values between schema tree elements are first computed using the specified element measures. These values are then ranked and the ones that are higher than a predefined *threshold (th)* are selected. The results of these evaluations are reported in Fig. 7.2. Figures 7.2(a,b) indicate that no single element measure is able to discover the qualitative correspondences. The name measure achieves F-measure ranging between 20% to 58%, the data type measure produces F-measures between 29% and 32%, while the cardinality constraint measure gives F-measure between 17% and 37%.



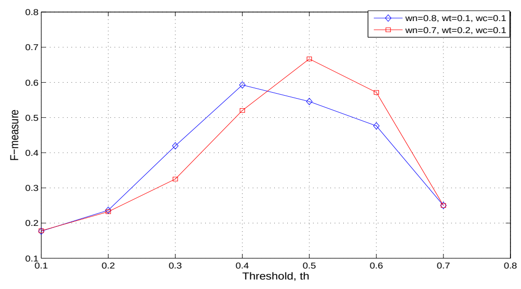
(a) Name measure



(b) Type and cardinality measures



(c) Name measure with one of other internal measures

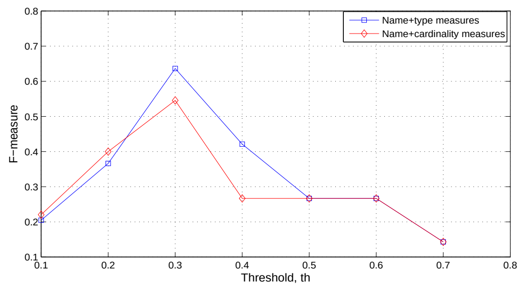


(d) Internal measures with different combinations

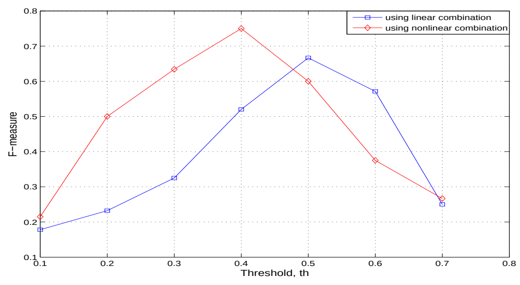
Figure 7.2: Internal measures quality using the linear combining strategy.

To get better matching quality, different combinations have been used. First, the name measure is combined with one of the other internal measures. The results reported in Fig. 7.2(c) show that combining name and type/data type measures performed better than the other combination. Then, the name measure is combined with the two other measures. Figure 7.2(d) illustrates that F-measure improves and its value reaches 67% when combining name, type, and cardinality constraints measures with weights $w_n = 0.7$, $w_t = 0.2$, and $w_c = 0.1$ for the name, data type, and cardinality constraint measures, respectively.

- Using nonlinear combination methods.
  The above set of experiments have been repeated using the nonlinear combining strategy with all internal measures. Further, these experiments are directed and guided by the results obtained from the first set. The results of these experiments are reported in Fig. 7.3. The figure presents two interesting findings. (1) The largest F-measure occurs at lower threshold values, as shown in Fig. 7.3(a), compared to the results obtained using the linear strategy,



(a) Name+ one of internal measures



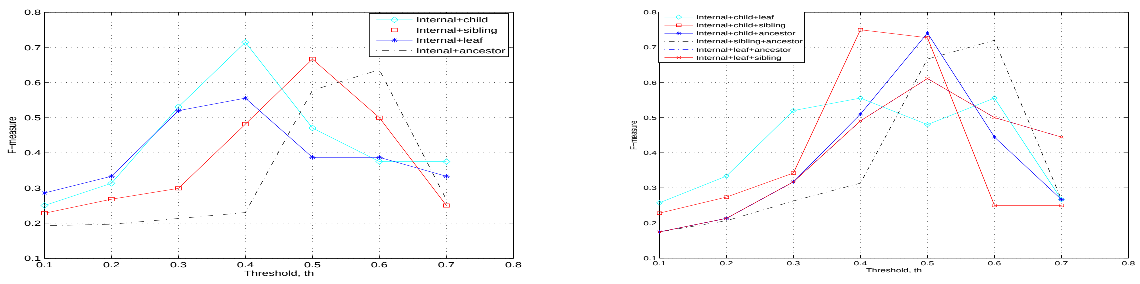
(b) Internal measure with nonlinear/linear combination

Figure 7.3: Internal measures quality using the nonlinear combination strategy.

as shown in Fig. 7.2(c). (2) Using the nonlinear combination strategy to aggregate a small number of element measures is not effective as using it to combine a large number of element measures. Figure 7.3(b) shows that using the nonlinear combining strategy achieves higher F-measure than the linear strategy. However, using the nonlinear strategy, as shown in Fig. 7.3(a), to aggregate the name measure with another element measure does not achieve higher quality.

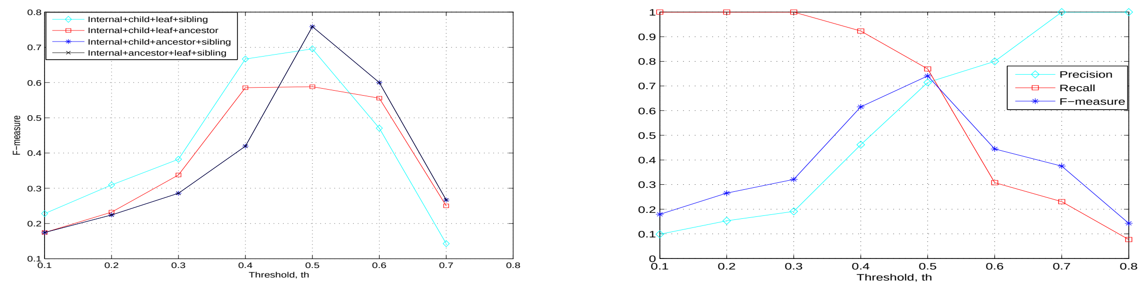
2. **Internal & external measures quality**.

   - Using the linear fixed combination method
   The second set of experiments was implemented to observe the quality of internal element similarity measure with different combinations of external element measures. This set of experiments has the same procedure as the first one: similarity values are first computed, ranked, and ones that are higher than the predefined *threshold (th)* are selected. The results of these evaluations are reported in Fig. 7.4. Combining the leaf context with the internal measure deteriorates the matching quality, as shown in Fig. 7.4(a), while the child context outperformed the other combinations. Figure 7.4(b) shows that combining also the child context with another element context other than the leaf context surpasses the other combinations, while Figure 7.4(c) gives the results of combining three of the external measures with the internal measure. Figure 7.4(d) outlines the results produced by combining the internal and external measures. The figure presents an interesting finding regarding to the used threshold (th). Small values of threshold result in a large number of false posi-



(a) Internal+one context measure.

(b) Internal+two context measures.



(c) Internal+three context measures.

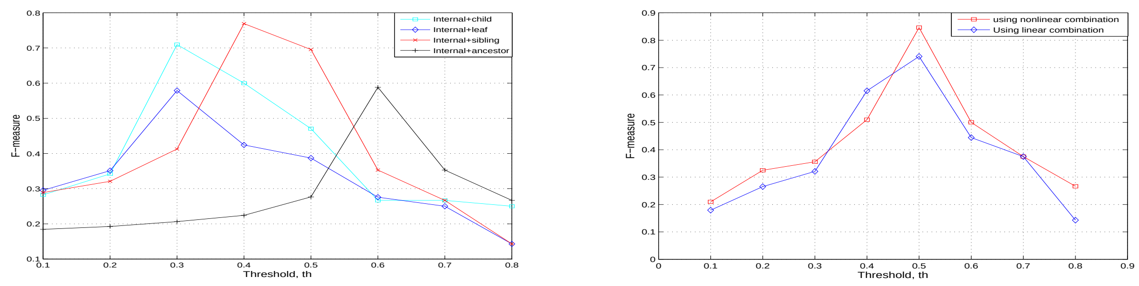
(d) Internal+ external measures.

Figure 7.4: Internal & external measures quality using the linear combining strategy.

tives (small precision values) and a small number of false negatives (large recall values). Increasing the value of threshold causes an opposite situation. The highest F-measure (0.76) was obtained at a threshold of 0.5. It should be noted that the highest F-measure using only the internal measure is 0.67 at the same threshold.

- Using the nonlinear combination method.
  We evaluated the quality of the external measures again using the nonlinear combining strategy. These set of experiments are also directed and guided by the results obtained from the set using the linear strategy. The results of these experiments are reported in Fig. 7.5. The figure presents two interesting findings. (1) The largest F-measure does not occur at lower threshold values, as shown in Fig. 7.5(a,b). Compared to the results using the linear strategy, largest F-measure occurs at the same threshold value. Indeed, combining the internal measure with external measures increases the element similarity that results in higher F-measure values at higher threshold values. (2) Using the nonlinear combination strategy achieves higher F-measure than the linear strat-

egy, as shown in Fig. 7.5(b). It should be noted that F-measure increase from 76% using the linear combining strategy to 84% using the nonlinear strategy at the same threshold.



(a) Internal with one of context measures.

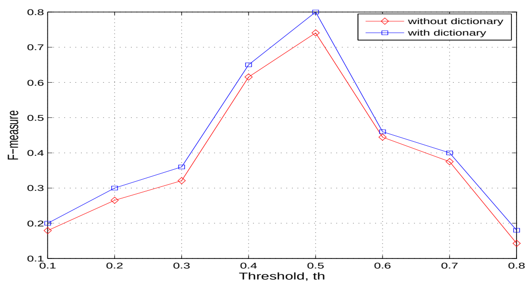
(b) Internal& external measure with nonlinear/linear combination

Figure 7.5: Internal & external measures quality using the nonlinear combination strategy.

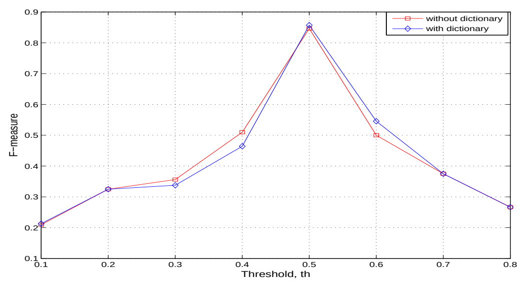
3. **Effect of external information sources**.

Although the tested schemas are small, matching is not of high quality due to different heterogeneities existing in them. F-measure values range between 17% and 76% depending on the used element measures and the selected threshold. To improve the matching quality, one method is to use semantic measures. To this end, we built a domain-specific dictionary, and we developed another set of experiments to observe the effect of external information sources on the matching quality. The results of these evaluations are reported in Fig. 7.6(a). The figure presents the effect of using an external dictionary on the matching quality using the linear combining strategy. Compared to results shown in Figure 7.4, F-measure has nearly the same value with/without the external dictionary at a threshold value of 0.1. At higher threshold values, F-measure has been improved gradually. It increases from 26% to 30% at a threshold value of 0.2, from 61% to 65% at 0.4, and from 76% to 80% at 0.5. The best F-measure obtained is 80% at a threshold of 0.5 using the external dictionary, and 76% without the dictionary.

To study the trade-off between matching quality and matching efficiency due to adding the domain-specific dictionary on the matching performance, we calculate



(a) using linear combining strategy.



(b) using nonlinear combining strategy

Figure 7.6: Effect of using external dictionary.

the quality improvement ratio (QIR) of using the external source as:

$$QIR = \frac{quality\,increment}{quality\,without\,external\,source} = \frac{4}{76} = 5.26\% \tag{7.8}$$

This quality improvement normally causes a decline in matching efficiency, computed as the time needed to perform the matching task. The element measures need 200 ms to complete the matching task without external dictionary, while they need 235 ms in the presence of the external dictionary. We calculate the performance decline ratio (PDR) as:

$$PDR = \frac{performance\,decrease}{performance\,without\,external\,source} = \frac{35}{200} = 17.5\% \tag{7.9}$$

This means that in order to improve the matching quality by a ratio of 5.26%, we must pay a performance cost ratio of 17.5%. Therefore, a trade-off between matching quality and matching efficiency should be considered, especially in the large-scale context.

In case of utilizing the nonlinear combining strategy, as shown in Fig. 7.6(b), the effect of using the external dictionary on the matching quality is not significant. It increases F-measure from 0.846 to 0.857 at a threshold of 0.5, which results in a quality improvement ratio of 1.3% ($\frac{85.7-84.6}{84.6} \times 100$) at the same performance decline ratio.

7.3.2 Lessons learned from the first scenario.

The experiments that we conducted in the first scenario present several interesting findings that can be used as a guide to develop experiments in the second scenario. These findings include:

- Using a single element similarity measure is not sufficient to assess the similarity between XML schema elements. This necessitates the need to utilize several element measures exploiting both internal element features and external element relationships.

- Utilizing several element measures provides the advantage of our matching algorithms to be more flexible. However, it also embeds a disadvantage of how to combine these similarity measures. We settle on selecting the aggregation function (weighted-sum) as a combining strategy making use of the two combining strategies: linear and nonlinear. According to the linear strategy, equations in Def. 5.5 and 5.8 can be written as follows

$$\begin{aligned} InterSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right) &= w_n \times Nsim\left(\mathcal{E}\ell_1.name, \mathcal{E}\ell_2.name\right) + \\ &\quad w_d \times Tsim\left(\mathcal{E}\ell_1.type, \mathcal{E}\ell_2.type\right) + \\ &\quad w_c \times Csim\left(\mathcal{E}\ell_1.card, \mathcal{E}\ell_2.card\right) \end{aligned}$$

$$\begin{aligned} ExterSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right) &= w_{ch} \times ChSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right) + \\ &\quad w_l \times LeafSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right) + \\ &\quad w_{sib} \times SibSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right) + \\ &\quad w_{an} \times PSim\left(\mathcal{E}\ell_1, \mathcal{E}\ell_2\right) \end{aligned}$$

  Reported results demonstrate that the name measure has the most effect of the internal measures, while external measures are nearly of equal effect. As a result, we set $w_n \gg (w_d \cong w_c)$, and $w_{ch} \cong w_l \cong w_{sib} \cong w_{an}$. Using the nonlinear combining strategy provides the possibility to consider the interdependencies between element similarities, and thus improves the matching quality compared to the linear strategy.

- Selecting the candidate correspondences is largely based on the value of *threshold*. Low values of threshold result in a large number of false positives (very low precision) and a small number of false negatives (high recall), while high values of threshold cause an inverse situation, as shown in Fig. 7.6. Therefore, we settle on medium values for thresholds ranging between 0.4 to 0.5.

- Exploiting external information sources, such as WordNet or domain-specific dictionaries, improves the matching quality. However, to get this improvement, the matching efficiency declines. In the large-scale context, a trade-off between matching effectiveness and matching efficiency should be considered. As a consequence, we decide not to exploit external information sources and we depend largely on our matching algorithms.

7.4 Second scenario: XPrüM evaluation

Our strategy in this scenario is to validate different element measures and their combinations using real-world data sets guided by the experimental results from the first scenario. Both the effectiveness and the efficiency of element measures have been evaluated as follows.

7.4.1 Data set

In this scenario, we experimented with data sets from 6 different domains. Schemas from the university domain are heterogeneous, i.e., they are developed independently, while schemas from the other five domains are homogenous, i.e. they are derived from their XML documents. From each domain, we collected four schemas. We choose the data sets because they capture different characteristics in the number of schema elements (schema size) and their depth (the number of node nesting), and they represent different application domains, as shown in Table 7.1.

Table 7.1: Data set details

Domain	No. of Schemas/elements	Avg. No. elements	Min./max. depth	total size (KB)
Article	4/530	135	5/10	100
bibliography	4/60	15	6/6	8
Car	4/344	83	5/6	30
Company	4/60	15	6/6	8
Movie	4/40	10	5/6	8
University	4/38	10	4/5	8

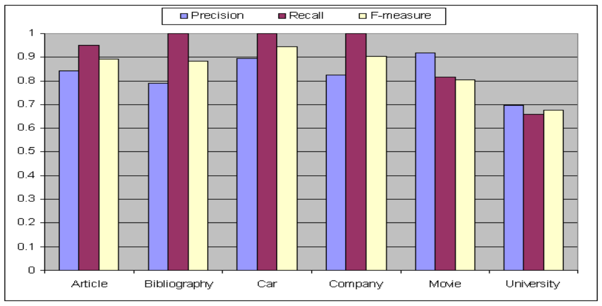
7.4.2 Evaluation methodology

Every schema pair inside the corresponding domain has been matched at a time. Hence, we have a total of 36 ($\frac{S\times(S-1)}{2} \times d$, where S is the number of schemas in each domain and d is the number of domains) matching tasks. The required parameters, such as the threshold value, are selected guided by the findings obtained from the first scenario. Furthermore, each matching task has been conducted utilizing both combining strategies. To evaluate the performance of element measures, we make use of the performance criteria mentioned above. The performance for each matching task is first evaluated and then matching tasks within the same domain have been averaged.

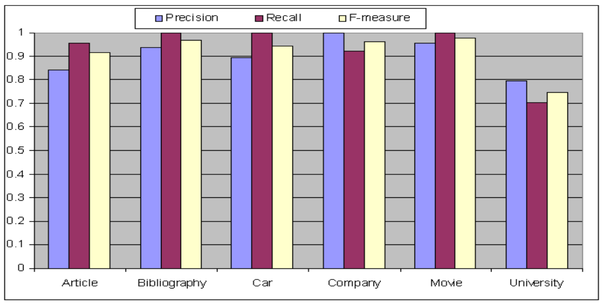
7.4.3 Experimental results

Quality evaluation. For each schema pair in the same domain, we conducted two sets of experiments—one using the linear combining strategy and the other using the nonlinear strategy. Element similarity measures (all matchers) discover candidate matchings that exceed the predefined threshold. The matching quality criteria are then computed for the schema pair. The quality criteria for all schema pairs in the same domain are then averaged to obtain the final quality criteria for the domain. Results are summarized in Figure 7.7, and present several interesting findings. (1) The nonlinear combining strategy outperforms the linear strategy across all the tested domains. Using the nonlinear strategy, F-measure ranges between 76% (the university domain) and 98% (the movie domain), while the linear strategy achieves F-measure between 68% (the university domain) and 95% (the car domain). This is due to the nonlinear strategy considers the interdependencies between element similarities. (2) Both combining strategies produce matching quality over the Article, Bibliography, Car, Company, and Movie domains higher than the matching quality over the University domain. This results due to the fact that XML schemas in the first set of domains are more homogeneous than XML schemas in the second set of domains.

Efficiency evaluation. To validate the efficiency of element measures, we collected more schemas from the 6 used domains. The number of schemas used in this set of experiments reaches 250 with a total of schema tree elements of 8000. The total size of tested schemas is 1.2 MB. Every schema pair in the data set (from different domains) is matched at a time. We determined the response time required to perform the matching task as a function of either the number of schemas or the number of elements.



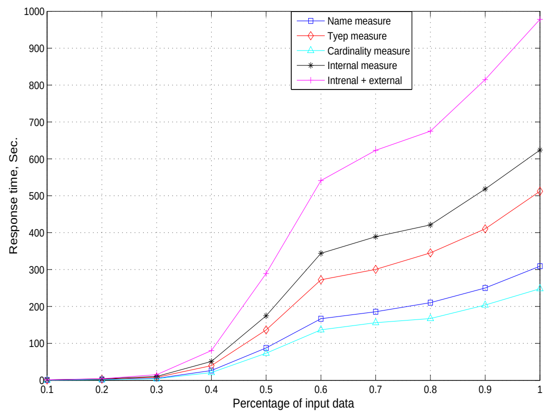
(a) Using the linear combination.



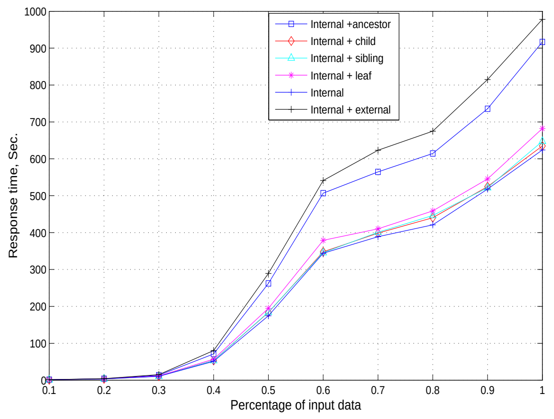
(b) Using the nonlinear combination.

Figure 7.7: Matching quality for real-world data sets.

Experimental results Figure 7.8 represents the results of the efficiency evaluation. The figure shows that the time complexity of the matching process is quadratic ($O(n^2)$, where n is the number of schema elements). The effect of the internal element measures on the matching efficiency is illustrated in Figure 7.8(a). The figure indicates that the cardinality constraint measure performs better than the other internal measures. This is due to the fact that the cardinality measure is based on small look up table. The figure also shows that while the type and constraint measures are based on similarity tables, however, the constraint measure is faster than the data type measure. This is due to the reason that the number of built-in XML data types is more than the number of cardinality constraints.



(a) Efficiency of internal measures.



(b) Efficiency of external measures.

Figure 7.8: Response time for real-world data sets.

Figure 7.8(b) presents the effect of the external element measures on the matching efficiency. The figure indicates that the ancestor context measures performs the worst among the other external measures. It can also be seen that child, sibling, and leaf context measures add insignificant response times to the matching efficiency. The reason behind this can be explained as follows: the ancestor measure is based on the path comparison, which requires the identification of paths to be compared. This identification process consumes much time.

7.4.4 More lessons learned.

The experimental results conducted in this study present several interesting findings that can be used as a guide during the development of schema matching. These findings include:

- It is evident that using a single element similarity measure is not sufficient to assess the similarity between XML schema elements. As the results in the first scenario indicate that no single measure is adequate to assess the similarity between element. The name measure has the strong effect among the internal measures on the matching quality, while, the child measure achieves the best matching quality among external measures when combining with the internal measure. This necessitates the need to utilize several element measures exploiting both internal features and external relationships of the elements. On the other hand, as the results from the second scenario states that the type measure is the most costly internal measure, while the ancestor measure is the most expensive external measure.

- Utilizing several element measures provides the advantage of the matching algorithms to be more flexible. However, it also embeds a disadvantage of how to combine these similarity measures. Furthermore, combining the internal measure with either the child or sibling context measure achieve better matching quality than combining the internal measures with the ancestor measure. The results in the second scenario also indicate that the ancestor measure consumes much time compared to the other external measures. They also confirm the fact that utilizing more measures results in better matching quality, however, with much response times.

- The strategy used to combine element similarity measures affects the matching quality, but it has no effect on the matching efficiency. According to results in the introduced scenarios, using the nonlinear combining strategy improves the matching quality.

7.5 Third scenario: EXPrüM evaluation

The set of experiments in this scenario is considered with analyzing the performance of the *EXPrüM* system in terms of *matching quality* and *matching efficiency* guided by the lessons learned from the first two scenarios. Moreover, to validate the performance of *EXPrüM* we compare it with two well schema matching systems, namely COMA++[1] [21, 48] and Similarity Flooding (SF) [101] as implemented in Rondo[2] [102].

7.5.1 Data sets

We experimented with the data sets shown in Table 7.2. These data sets were obtained from[3]. We choose them because they capture different characteristics in the numbers of nodes (schema size) and their depth (the number of nodes nesting), and they represent different application domains. We utilized two different data sets depending on the measured performance criterion: matching effectiveness or matching efficiency. The first set, Table 7.2 Part(A), is used to evaluate matching effectiveness, wherein schemas from the *TPC_H* and *bioinformatics* domains do not contain complex matches, while schemas from the other two domains contain complex matches. Data sets described in Table 7.2, *Part (B)*, are used to validate matching efficiency.

Table 7.2: Data set details

Part (A) Effectiveness					*Part (B) Efficiency*		
	TPC_H	Bibliography	Auction	Bioinformatics	Domain	No. of Schemas/nodes	Schema size
No. nodes(S_1/ S_2)	43/17	22/28	38/37	101/69	University	44/550	< 1KB
Avg. No. nodes	30	25	38	85	XCBL	570/3500	< 10 KB
Max. depth (S_1/ S_2)	3/6	6/7	3/4	6/6	OAGIS	4000/3600	<100 KB
					OAGIS	100/65000	>100 KB

Using these data sets, we defined 8 match tasks divided into two groups. The first group contains 4 match tasks, each matching two different schemas from data sets described in Table 7.2, *Part (A)* to validate the matching effectiveness. For each task, we

[1]http://dbs.uni-leipzig.de/Research/coma.html
[2]http://infolab.stanford.edu/ melnik/mm/rondo/
[3]

- http://www.cs.toronto.edu/db/clio/testSchemas.html
- http://sbml.org/Documents/Specifications/XML_Schemas
- http://www.xcbl.com
- http://www.oagi.org

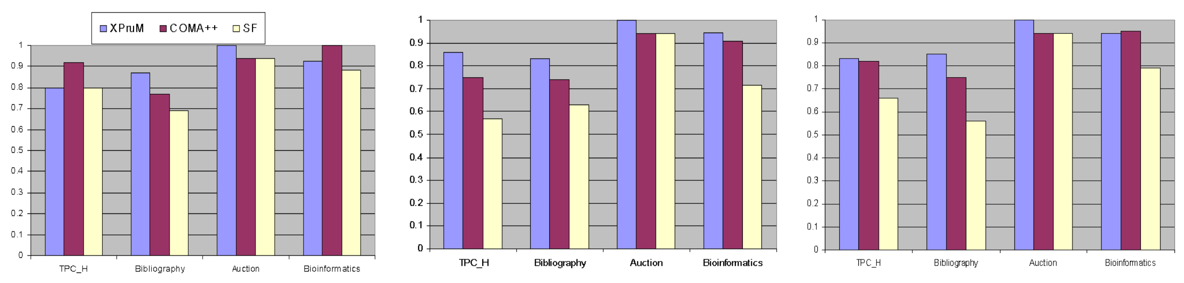
manually derived the real (correct) matches. The second group contains 4 match tasks, each matching a set of schemas from the same domain utilizing data sets described in Table 7.2, *Part (B)* to validate the matching efficiency.

7.5.2 Matching quality

The data set Part (A) illustrated in Table 7.2 is introduced to the *EXPrüM* system two schemas at a time. For each domain, we performed two experiments—from $S1$ to $S2$ and from $S2$ to $S1$. Matcher algorithms discover candidate matchings that exceed a predefined threshold. Then, these candidates are ranked for each element up to the top-3 ranking (if found). Finally, matching quality measures are computed. We also computed the matching quality of both the COMA++ system and the Similarity Flooding SF (RONDO) system and we compared them to our system. The results are summarized in Fig. 7.9.

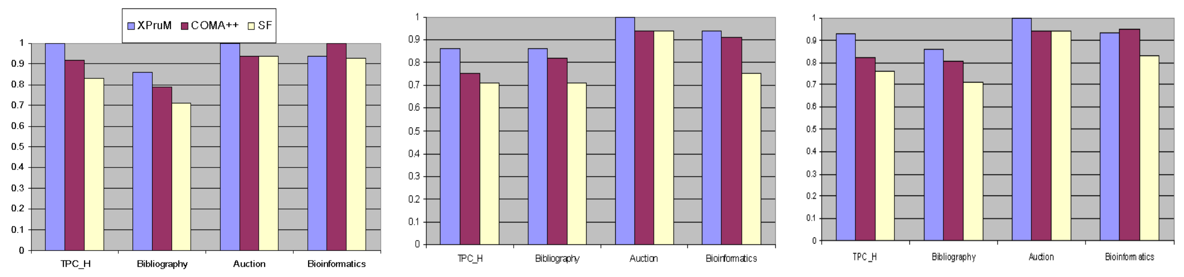
The results show that *EXPrüM* achieves high matching quality in terms of precision, recall, and F-measure across all four domains ranging from 80% to 100%. Compared to COMA++, which is mostly independent from the match direction (from $S1$ to $S2$ or from $S2$ to $S1$), our system, like SF, depends on the match direction. Figure 7.9 illustrates that matching quality measures for COMA++ using the *TPC_H, Auction* and *Bioinformatics* domains are the same from $S1$ to $S2$ (Fig. 7.9(a)) and from $S2$ to $S1$ (Fig. 7.9(b)). However, this is not true for the bibliography domain. The reason is that schemas from the bibliography domain contain more complex matches, which are harder to discover. As shown in Fig. 7.9, our system, which is able to cope with complex matches, achieves higher precision, recall, and F-measure than both *COMA++* and *SF* across the bibliography domain. The best matching results for *EXPrüM* are achieved from the *Auction* domain that includes less semantic and structural heterogeneities. We wish to remark that our system can identify all matchings including complex ones, whereas both *COMA++* and *SF* only identify element-level matchings with F-measure of 94%.

Individual matchers effectiveness. For each domain, we performed a set of experiments to study the effect of individual matchers on the whole matching quality. To this end, we considered the following combinations of matchers: (1) the name matcher alone, (2) the name matcher with the data type compatibility, and (3) the name matcher with the data type compatibility and the structural context matcher (i.e., the complete EXPrüM system). We use *precision* as a matching quality measure. Figure 7.10 shows the matching quality for these scenarios.



Precision Recall F-measure

(a) Computed from S1 to S2



Precision Recall F-measure

(b) Computed from S2 to S1

Figure 7.9: Matching quality measures for XPrüM, COMA++, and SF systems.

Figure 7.10 clearly shows the effect of each individual matcher on the total matching quality. The name matcher alone has very low accuracy on the first domain (10%), because the two schemas $S1$ and $S2$ present names with many slight variations and the name matcher utilizes very simple string similarity functions. Some more accurate results have been achieved for the other two domains (21.5% on the bibliography domain and 26% on the auction domain). Using the data type compatibility matcher with the name matcher provides an irrelevant improvement of matching accuracy (between 4% to 10%). In contrast, the best matching results of matcher combinations are achieved by adding the structural context matcher. This matcher improves matching precision by approximately 64%. Finally, we choose precision as a measure for matching quality in this scenario, since precision quantifies efforts needed to remove false positive candidates.

Matching quality for complex matchings. As stated before, we select the tested data set to reflect different features and characteristics. The *TPC_H* domain does not contain any complex matching and it is suitable for element-level matchings. The *bibliography* domain contains 4 complex matchings out of 20 total matchings. When performing match from $S1$ to $S2$, *EXPrüM* could correctly identify 2 out of 4 producing a precision of 50%. While performing match from $S2$ to $S1$, the system could correctly

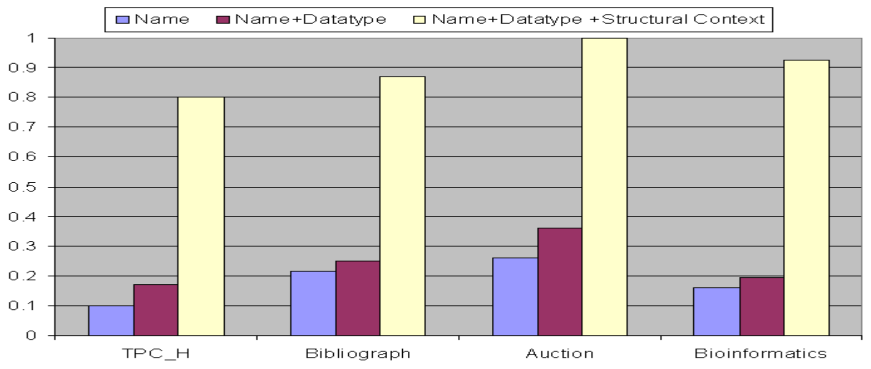


Figure 7.10: Matching precision for different combinations of matchers.

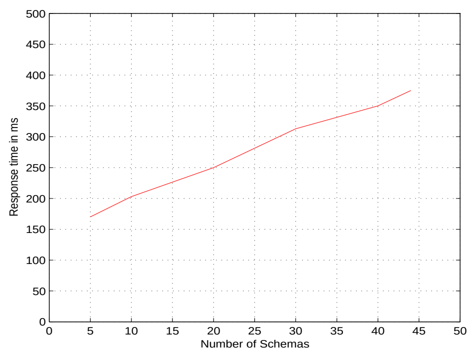
identify 3 out of 4 producing a precision of 75%. The third domain, the *Auction* domain, contains 5 complex matchings out of 32 total matchings. *EXPrüM* could identify all complex matchings along two matching directions, i.e., from $S1$ to $S2$ and from $S2$ to $S1$.

7.5.3 Matching efficiency

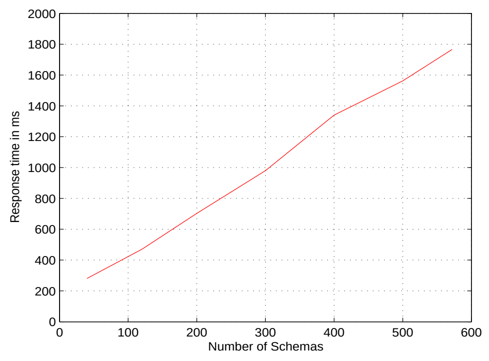
We measure the time response of our matching system as a function of the number of schemas and nodes through the data set Part (B) illustrated in Table 7.2. For schemas whose sizes are less than $10KB$, the matching response time is a function of the number of schemas. Otherwise, the response time is measured as a function of the number of nodes. Results are summarized in Figure 7.11.

Figure 7.11 shows that *EXPrüM* scales well across all three domains. The system could identify and discover correspondences across 44 schemas of 550 nodes from the university domain in a time of 0.4 seconds, while the approach needs 1.8 seconds to match 570 schemas with approximately 3500 nodes from the XCBL domain. This demonstrates that *EXPrüM* is scalable with a large number of schemas. To demonstrate the scalability of the system with large-scale schemas, we carried out two other sets of experiments. First, we considered the OAGIS domain that contains schemas whose sizes range between $10KB$ and $100KB$. Figure 7.11(c) shows that the system needs 26 seconds to match 4000 schemas containing 36000 nodes. Then, in the same domain, we considered 100 schemas whose sizes are larger than $100KB$. *EXPrüM* requires more than 1000 seconds to match 65000 nodes, as shown in Figure 7.11(d).

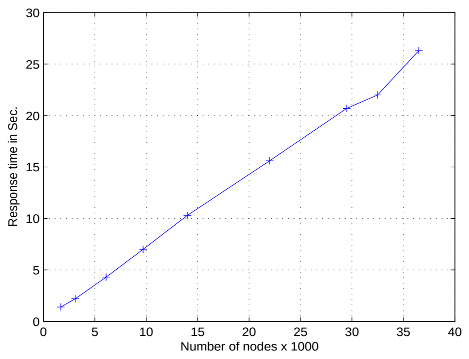
Effect of individual matchers on matching efficiency. In this subsection, we discuss the effect of the individual matcher combination on the matching efficiency. To this



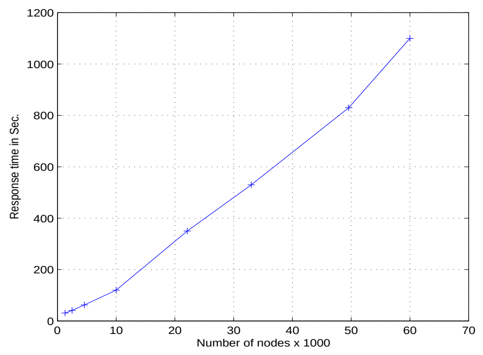
(*a*) Response time of University schemas.



(*b*) Response time of XCBL schemas.



(*c*) Res. time of OAGIS schema nodes 1.



(*d*) Res. time of OAGIS schema nodes 2.

Figure 7.11: Performance analysis of EXPrüM system with real-world schemas.

end, we performed a set of experiments by using the OAGIS domain with sizes ranging between $10KB$ and $100KB$ for the following scenarios: (1) the name matcher alone, (2) the name matcher with the data type compatibility, and (3) the name matcher with data type compatibility and the structural context matcher (i.e., the complete XPrüM system). Figure 7.12 shows the matching response time for these scenarios.

The results show that the name matcher needs less time than the other combinations, as it was expected. This matcher takes 16 seconds to match 36000 nodes. Adding the data type compatibility matcher increases the response time to 23 seconds, with an associated matching quality improvement ranging between 4% to 10%. It is interesting that by adding the structural context matcher it needs only 3 seconds more to perform the matching process, with an associated matching quality improvement of approximately 63%.

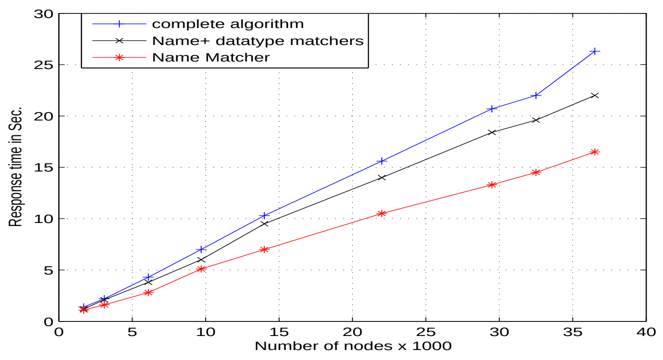


Figure 7.12: Matching response time for different combinations of matchers.

Matching quality/matching efficiency cost ratio

In order to evaluate the benefits behind the structural context matcher, we compute the ratio between matching quality improvement and matching efficiency cost. This ratio could be used to evaluate different combinations of matchers, and is denoted by $\eta_{matcher}$. The data type matcher is evaluated as follows:

$$\eta_{datatype} = \frac{MQI}{MEC} = \frac{10}{30} = 0.33 \tag{7.10}$$

where MQI, Matching Quality Improvement, is the incremental value of matching quality because of adding the data type matcher, and MEC, Matching Efficiency Cost, is computed by computing the percentage of increasing response time due to adding the matcher (i.e. $30 = \frac{23-16}{23} \times 100$). The structural matcher is computed as follows:

$$\eta_{structural} = \frac{MQI}{MEC} = \frac{63}{11} = 5.7 \tag{7.11}$$

Equations 7.10 and 7.11 show that the relative performance benefits from our new structural matcher. Although the match achieves 63% improvement in matching quality, this requires only a matching efficiency cost of 11%

7.5.4 Cost-effectiveness comparison

In addition to the above two scenarios, we evaluated our approach by comparing the cost-effectiveness of it with two recently well-known schema matching systems, namely COMA++ and PORSCHE. The two systems share our system in some features, including

Table 7.3: Cost-effectiveness data set details from [59]

	University	*Biology*
No. nodes(S_1/ S_2)	18/18	719/80
Avg No. nodes	18	400
Max. depth (S_1/ S_2)	5/3	7/3
No. mappings	15	57

that they are schema-based approaches; they utilize rule-based algorithms; and they accept XML schemas as input. However, they produce element-level mappings (one-to-one); they need pre-match effort, e.g., tuning match parameters and defining match strategy; they evaluate matching effectiveness using precision, recall, and F-measure (overall).

To conduct this set of experiments, we used two sets of XML schemas, each containing two schemas. These schemas are described and used in [59]. To make this work self-contained, we summarize the properties of the data sets in Table 7.3. The first one describes university courses and is used in the small-scale context, while the second one comes from the biology domain, and is used in the large-scale context.

Small-scale schemas: The cost-effectiveness of test matchers using small-scale schemas, the university domain, can be computed by the following equations:

$$CE_{COMA++s} = w_{OV} * OV_{COMA++} + w_t * \frac{1}{T_{COMA++n}} \tag{7.12}$$

$$CE_{Por_s} = w_{OV} * OV_{Por} + w_t * \frac{1}{T_{Por_n}} \tag{7.13}$$

$$CE_{EXPruMs} = w_{OV} * OV_{EXPruM} + w_t * \frac{1}{T_{EXPruMn}} \tag{7.14}$$

where OV_{COMA++}=0.53, OV_{Por}=0.67, OV_{WXPruM}=0.75 T_{COMA++}=0.9 s, T_{Por}=0.6s, T_{EXPruM}=0.188s and w_{OV}=0.8 (for small-scale schemas), and w_t=0.2, then

$$CE_{COMA_s}=0.424 + \tfrac{0.2}{1}=0.624,$$
$$CE_{Por_s}=0.536 + \tfrac{0.2}{\frac{0.6}{0.9}}=0.836 \text{ , and}$$
$$CE_{EXPruM}=0.6 + \tfrac{0.2}{\frac{0.188}{0.9}}=1.56$$

Table 7.4: Summary of cost-effectiveness comparison results

Evaluated System	OV		T		CE	
	small-scale	large-scale	small-scale	large-scale	small-scale	large-scale
COMA++	0.53	0.4	0.9s	4s	0.624	0.64
PORSCHE	0.67	0.3	0.6s	2s	0.836	0.98
EXPrüM	0.75	0.6	0.188s	1.87s	1.56	1.25

Large-scale schemas: The cost-effectiveness of test matchers using large-scale schemas, the biology domain, can be computed by the following equations:

$$CE_{COMA++_l} = w_{OV} * OV_{COMA++} + w_t * \frac{1}{T_{COMA++_n}} \tag{7.15}$$

$$CE_{Por_l} = w_{OV} * OV_{Por} + w_t * \frac{1}{T_{Por_n}} \tag{7.16}$$

$$CE_{EXPruM_l} = w_{OV} * OV_{EXPruM} + w_t * \frac{1}{T_{EXPruM_n}} \tag{7.17}$$

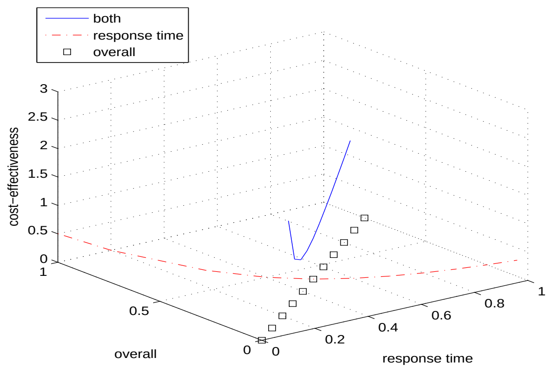
where OV_{COMA++}=0.4, OV_{Por}=0.3, OV_{XPruM}=0.6, T_{COMA++}=4s, T_{Por} =2s, T_{XPruM}=1.8 and w_{OV}=0.6 (for large-scale schemas), and w_t=0.4, then

$$CE_{COMA_l}=0.24 + \frac{0.4}{1}=0.64,$$
$$CE_{Por_l}=0.18 + \frac{0.4}{\frac{2}{4}}=0.98,$$
$$CE_{EXPruM_l}=0.36 + \frac{0.4}{\frac{1.8}{4}}=1.25$$

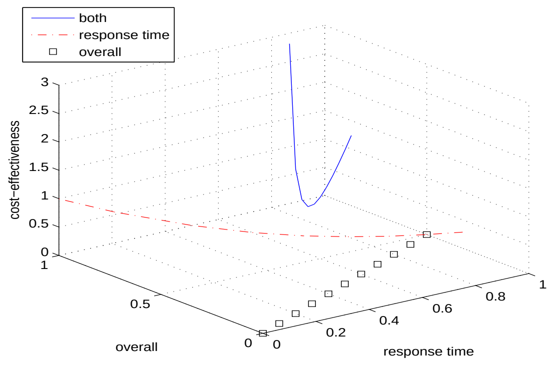
Discussion.

In this section, we conducted a comparison between our proposed system and two recently well-known systems, *COMA++* [48] and *PORSCHE* [122], considering both performance aspects: matching effectiveness and matching efficiency. The three systems have been evaluated using small-scale schemas (university schemas) and large-scale schemas (order schemas) and their cost-effectiveness have been computed. Table 7.4 reports these results. We observed that *EXPrüM* outperforms over the other two systems in both cases, small-scale and large-scale. This observation can be explained, as *EXPrüM* exploits the semantic and structure information of schema trees in an efficient way that provides high matching quality and matching efficiency.

We studied the relationship between cost-effectiveness and both performance aspects (overall and response time). Figure 7.13 illustrates this relationship, where the squared line only represents the overall, the dashed line represents the response time, and the



(a) small-scale schemas



(b) large-scale schemas

Figure 7.13: Performance aspects with cost-effectiveness.

solid line represents both. Figure 7.13(a) is drawn for the small-scale case (i.e.w_{OV} $=.8$ and w_t=$.2$) while Fig. 7.13(b) is drawn for the large-scale schemas (w_{OV} $=.5$ and w_t $=.5$). In case of small-scale schemas, the cost-effectiveness is more biased to the overall measure than the response time of the system, while in case of large-scale schemas, the cost-effectiveness is biased by both performance aspects.

7.6 Summary

In this chapter, we presented the experimental evaluation that validated the performance of our proposed system. To realize our goals, we have three different scenarios. The first is to evaluate XML element similarity measures with different combination strategies to extract lessons that can be used as a guide during conducting experiments in the other scenarios. Experiments in the second scenario have been used to validate the performance of our system, *XPrüM* utilizing real-world data sets. The third scenario is devoted to evaluate the performance of *EXPrüM* and to compare it with two well-known schema matching systems. The results are encouraging and empirically prove the strength of our approach. Finally, we have conducted another set of experiments to consider the cost-effectiveness measure, a new measure that we introduced to consider both performance aspects at the same time.

Experimental results have shown that *XPrüM/EXPrüM* scales well in terms of both large numbers of schemas and large-scale schemas. Moreover, it can preserve matching quality considering both simple and complex matching. We have introduced and measured the matching quality improvement/matching efficiency cost ratio to validate our new structural matcher.

Our system includes other features: it is almost automatic; it does not make use of any external dictionary or ontology; moreover, it is independent from data models and application domains of matched schemas. In the following chapters, we show how to deploy our sequence-based matching approach in several applications and domains, such as XML data clustering and Web service discovery.

Part IV

Deployment of the proposed schema matching system

8

SeqXClust: XML schema clustering framework

XML is emerging as a de facto standard for information representation and exchange on the Web and within organizations. There has been a growing need to develop high-performance techniques that manage large XML data repositories efficiently. A nice and elegant solution is to group similar XML data based on their content or their structure or both. The process of grouping similar XML data is called XML data clustering.

The relationship between XML data clustering and schema matching is bidirectional. On the one side, clustering techniques have been adopted to improve the matching performance, and on the other side schema matching is a fundamental step to clustering techniques. In this chapter, we aim to deploy our sequence-based schema matching approach developed and implemented in the previous chapters. Particularly, we utilize the approach to compute the similarity between XML data as a guide to cluster them. We first introduce the XML data clustering process demonstrating the need to new approaches. We then present a novel XML schema clustering framework, called *SeqXClust*, that is based on our schema matching algorithms. To validate the proposed framework, we conducted a set of experimental evaluations showing that our framework is accurate and scales well in clustering large and heterogeneous XML data.

The material presented in this chapter has been developed and published in [13, 7].

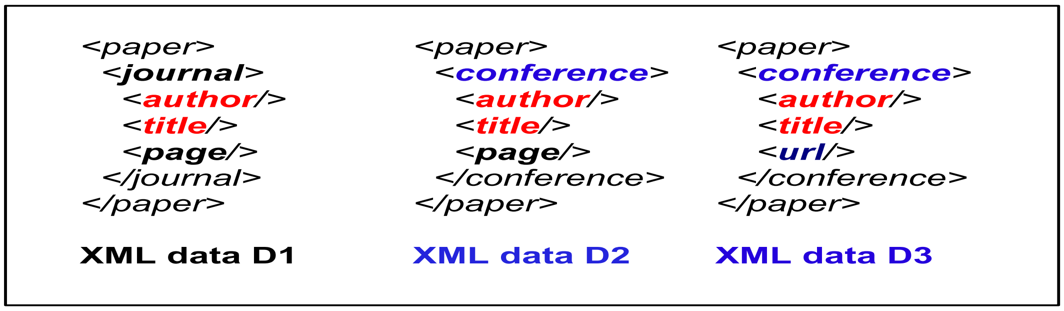
8.1 Introduction

Due to XML's inherent features, XML is emerging as a standard for information representation and exchange among various applications on the Web and within organizations. As a result, a huge amount of information is formatted in XML data and several tools have been developed to deliver, store, integrate, and query XML data [32, 71]. There has been a growing need for developing high-performance techniques to manage and analyze these giant XML data efficiently. In order to do this, a possible and elegant solution is to group similar XML data according to their content or their structures or both. Grouping similar XML data across heterogeneous ones is known as *XML data clustering*.

Commonly, clustering is a useful technique for grouping data objects, such that objects within a single group/cluster have similar features, while objects in different groups are dissimilar [82, 28]. There are two types of XML data—*XML documents* and *XML schemas*, as stated in Chapter 2. An XML schema is the description of the structure and the legal building blocks for an XML document. Several XML schema languages have been proposed [88]. Among them, XML DTD and XML Schema Definition (XSD) are commonly used. An XML document (document instance) represents a snapshot of the content of the XML document, since the document definition outlined in a schema holds true for all document instances of that schema. Therefore, the result produced from the clustering of schemas will hold true for all document instances of those schemas and can be reused for any other instances. On the contrary, the result of clustering of document instances will only hold true for included document instances [108]. The clustering process should be repeated for new document instances. *Therefore, in this chapter, we only consider the clustering of XML schemas.*

Clustering XML data is an intricate process and differs significantly from clustering of flat data and text. The difficulties of clustering XML data are due to several reasons [3]. Among them are:

- Clustering algorithms require the computation of similarity between different sets of XML data, which is itself a difficult research problem (the heterogeneity in XML data presents many challenges to identify the ideal similarity function). For example, Figure 8.1 shows three XML data representing journal and conference papers in the DBLP database. The data sets have common elements such as *author* and *title*. Even if D1 and D2 have only one different element, they should be in two different clusters according to usual semantics that give different relevance to publications



```
<paper>
  <journal>
    <author/>
    <title/>
    <page/>
  </journal>
</paper>
```

XML data D1

```
<paper>
  <conference>
    <author/>
    <title/>
    <page/>
  </conference>
</paper>
```

XML data D2

```
<paper>
  <conference>
    <author/>
    <title/>
    <url/>
  </conference>
</paper>
```

XML data D3

Figure 8.1: Different XML data

in journals and conferences. In contrast, even if D2 and D3 have only one different element, they should be in the same cluster because they refer to conference papers.

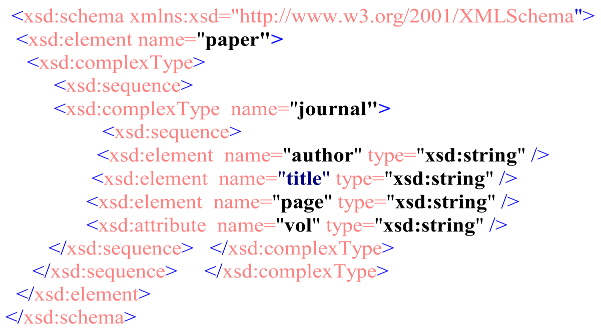
- The structural organization of the XML data increases implicit dimensionality that a clustering algorithm needs to handle, which leads to meaningless clusters.

Research on clustering XML data is gaining momentum to address these challenges [108, 106, 92, 94, 3, 89] both for clustering XML documents and clustering XML schemas. Motivated by the above challenges, in this chapter, we present a new schema matching-based approach to XML schema clustering. The work in this chapter presents a novel methodology that quantitative determines the similarity between XML schemas by considering both semantic and structural features of XML data. This work enhances XML data clustering by representing XML schemas as sequence representations utilizing the Prüfer encoding method [115]. This representation improves clustering solution quality as well as clustering solution performance. We carried out a set of experiments utilizing different real data sets to evaluate the proposed framework. Our experimental results show that the proposed framework is fast and accurate in clustering heterogenous XML data.

Before detailing the proposed clustering framework, we present a brief description of XML data clustering in general. As stated in Chapter 3, XML data clustering activity typically involves three phases [82, 28].

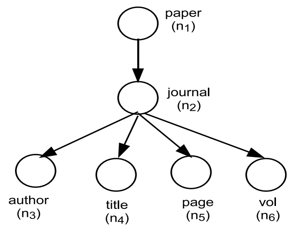
8.1.1 Data representation

In this phase, the focus is to represent the XML data using a common data model such as rooted labeled trees, directed acyclic graphs, or vector-based techniques as well as to identify what to consider in the next phase—similarity computation. This common model



```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="paper">
  <xsd:complexType>
     <xsd:sequence>
     <xsd:complexType  name="journal">
          <xsd:sequence>
         <xsd:element  name="author" type="xsd:string" />
        <xsd:element  name="title" type="xsd:string" />
       <xsd:element  name="page" type="xsd:string" />
       <xsd:attribute  name="vol" type="xsd:string" />
    </xsd:sequence>  </xsd:complexType>
  </xsd:sequence>    </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

(a) XML schema of D1



(b) Data tree

Figure 8.2: Tree representation of XML schema of D1

should capture both content and structure features of XML data. Data representation starts with parsing XML data using an XML parsing tool such as the SAX parser[1]. In case of XML schema clustering, the parsing process may be followed by a normalization process to simplify the schema structure according to a series of predefined transformation procedures similar to those in [89]. The commonly used method to represent XML data is *labeled trees*, such as trees defined in Definition 5.1, (XML data can be represented as data tree. A *data tree* (DT) is a rooted labeled tree defined as a 3-tuple $DT = (N_T, E_T, Lab_{NT})$).

Figure 8.2 illustrates a data tree of an XML schema of the XML data $D1$ represented in Fig. 8.1. Each node in the data tree is associated with the *name* label, such as "paper" and "journal" as well as its OID, such as n_1 and n_2. The nodes n_1, n_2 and n_3 represent examples of the element nodes, while node n_6 is an attribute node. A data tree DT is called an *ordered labeled tree* if a left-to-right order among siblings in DT is given, otherwise it is called *unordered tree*.

8.1.2 Similarity computation

The main aim of the similarity computation phase is to assess and determine the similarity between XML data exploiting their elements' features and/or relationships among them identified in the data representation phase. There are several methodologies to perform this task depending on the used data representation [134].

[1]http://www.saxproject.org

Tree-based similarity approaches

The computation of similarity among XML data represented as data trees depends on the exploited elements on which similarity function are applied. Based on the exploited elements, we classify the similarity measures into: *element-level measures* and *tree-level measures*.

Element-level measures, also known as *schema matching-based* methods, consider element details either as internal features or external relationships, such as the element similarity measures described in Chapter 5. In element-level measures, the similarity between XML data is based on the computed similarities among the elements of the schema.

On the other hand, *tree-level measures*, also known as *tree-editing* methods, exploit complex objects without taking into account the detailed object components in the data tree. The tree-editing problem is the generalization of the problem of computing the distance between two strings to labeled trees. As usual, the edit distance relies on three elementary edit operations: the *relabeling*, which consists of replacing a label of a node by another label, the *insertion* of a node, and the *deletion* of a node.

Let DT_1 and DT_2 be two data trees. The edit distance between DT_1 and DT_2, denoted $\delta(DT_1, DT_2)$, is the minimal cost of edit operations needed to transform DT_1 into DT_2, that is:

$$\delta(DT_1, DT_2) = min\{\gamma(S) | S\ is\ an\ edit\ operation\ sequence\ transforming\, DT_1\, to\, DT_2\} \tag{8.1}$$

Figure 8.3 illustrates that the edit operations required to transform $DT1$ to $DT2$ equal to those required to transform $DT2$ to $DT3$, because only one relabeling operation is required in both cases to transform the source tree into the target tree. A dotted line from a node in a data tree, such as $DT1$, to a node in another data tree, such as $DT2$, indicates that a relabeling operation is required. Assigning a constant cost for the edit operations results in an equal tree distance between $DT1$ and $DT2$ and $DT2$ and $DT3$. This simple example shows that the tree editing method may not be able to distinguish the hierarchical difference in some situations. To overcome this problem, as we will show below, both semantic and structural features of data trees should be exploited. Moreover, computing tree edit distances turns out to be expensive, as it requires a quadratic number of comparisons between data tree elements. That makes using the tree-editing method to compute the similarity among XML data unpractical, especially when dealing with large

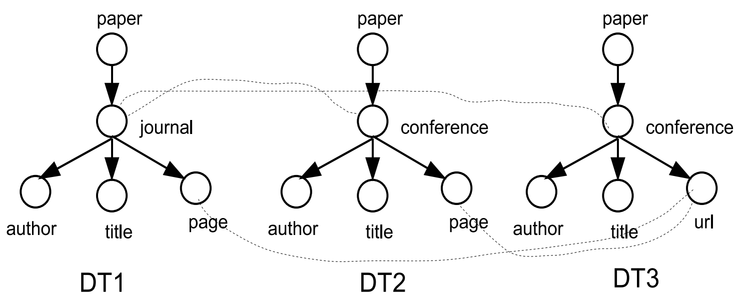


Figure 8.3: Tree distance between XML data

XML data sources.

8.1.3 Clustering/grouping

This phase proceeds for data mining. XML data that are similar in structures and semantics are grouped together to form a cluster using a suitable clustering algorithm [82].

Clustering methods are generally divided into two broad categories. *Non-hierarchical methods* group a data set into a number of clusters using a pairwise distance matrix that records the similarity between each pair of documents in the data set. *Hierarchical methods* produce nested sets of data (hierarchies), in which pairs of elements or clusters are successively linked until every element in the data set becomes connected. Non-hierarchical methods have low computational requirements, but certain parameters like the number of formed clusters must be known a priori. Hierarchical methods are computationally expensive. However, these methods have been used extensively as a means of increasing the effectiveness and efficiency of retrieval. For a wide-ranging overview of clustering methods one can refer to [82, 28].

Despite the widespread use, the performance of both hierarchal and non-hierarchal clustering solutions decreases radically when they are used to cluster large scale and/or large number of XML data. Therefore the need to another set of clustering algorithms arises. Among them are the incremental clustering [106] and constrained hierarchal agglomerative algorithms [80].

8.2 SeqXClust: The clustering framework

Our proposed approach is based on the exploitation of the structure and semantic information from XML schemas. The objective is to deploy our sequence-based matching

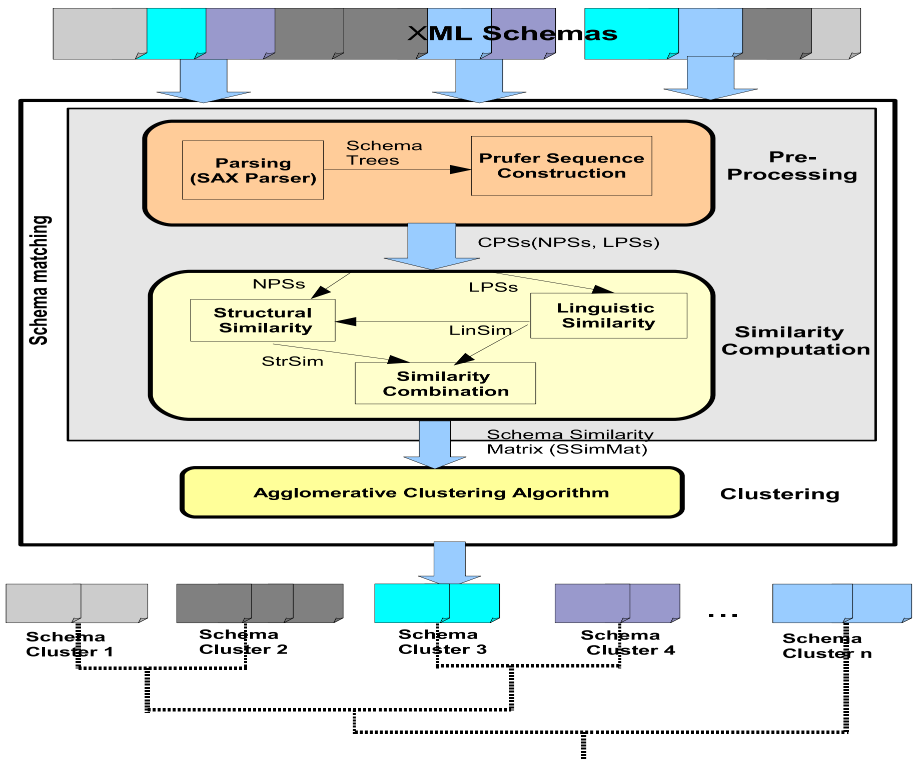


Figure 8.4: XML clustering framework architecture

approach that assesses the similarity between XML schemas. The measured similarity is then used as a guide to group similar XML data.

To realize this goal, we have developed and implemented an XML schema similarity assessment framework, called *SeqXClust*. The framework, as shown in Fig. 8.4 and inspired from data clustering activity phases, consists of three main phases: *Pre-processing*, *similarity computation*, and *clustering*. The Pre-processing phase is considered with the representation of XML schemas as schema trees and then the extension of schema trees to sequence representations using the Prüfer encoding method. The sequences should capture both semantic and structure information of schema trees. The similarity computation phase aims to assess the similarity between XML schemas exploiting both information to construct a schema similarity matrix. Then, the clustering algorithm makes use of the similarity matrix to group the similar XML data. As shown in Fig. 8.4, the first two phases constitute our schema matching approach.

The outline of the algorithm implementing the proposed framework is shown in *Algorithm* 10. The algorithm accepts a set of XML schemas as input, $S = \{S_1, S_2, ..., S_n\}$, to group the similar XML schemas according to their structure and semantics. To this, we first analyze each XML schema and represent it as a data tree, *line* 2, using a SAX parser. Each data tree is then examined and represented as a sequence representation, CPS, *line* 3. The algorithm proceeds to compare all CPS pairs to assess the similarity between them using our developed sequence matching algorithms, *line* 11. The returned similarity value is stored in its corresponding position in the schema similarity matrix, $SSimMat$. Finally, the similarity matrix will be exploited by a suitable clustering algorithm, *line* 15, to produce the cluster set, $Clust\_Set$.

In fact, this outline indicates that the XML schema clustering framework has common aspects with our developed schema matching techniques. Therefore, in the following section, we shed light on two specific aspects: *schema similarity matrix construction* and *used clustering algorithms.*

Algorithm 10: XML schema clustering algorithm

```
input  : A set of XML schemas, $S = \{S_1, S_2, ..., S_n\}$
output: A set of clusters, $Clust\_Set$
1  for $i \leftarrow 1$ to $n$ do
2      $DT[i] \leftarrow$ buildDT $(S_i)$;
3      $CPS[i] \leftarrow$ buildCPS $(DT[i])$;
4  end
5  $SSimMat[][] \leftarrow 0$;
6  for $i \leftarrow 1$ to $n-1$ do
7      $CPS_i \leftarrow (CPS\ [i])$;
8      $MatchRes \leftarrow 0$;
9      for $j \leftarrow i+1$ to $n$ do
10         $CPS_j \leftarrow (CPS\ [j])$;
11         $MatchRes \leftarrow$ schemaMatch $(CPS_i, CPS_j)$;
12         $SSimMat[i][j] \leftarrow average\ (MatchRes)$;
13     end
14 end
15 $Clust\_Set \leftarrow$ cluster $(SSimMat)$
```

8.2.1 Schema similarity matrix

The first two phases, *Pre-processing* and *Similarity Computation*, have the same steps and details as our schema matching approaches, as shown in Fig. 8.4. Therefore, we consider only how to construct the output of these two phases. We name the output of these phases as *Schema Similarity Matrix, SSimMat.* The problem of constructing $SSimMat$

can be stated as follows. Given a set of XML schemas, $S = \{S_1, S_2, ..., S_n\}$, construct an $n \times n$ schema similarity matrix.

After the Pre-processing phase that constructs corresponding CPSs from XML schemas, as shown in *Algorithm* 10, the algorithm starts by initializing matrix elements, *line* 5. Then, it processes individual CPS pairs in the data set, CPS_i and CPS_j, *lines* 7& 10. For each pair, the algorithm calls our schema matching algorithm, *line* 11, to identify and determine the semantic correspondences between every schema pair and stores them in a matching result list, $MatchRes$. Each item in the list represents a mapping element. To get a similarity value between two schemas, the algorithm uses an average function that sums up all element similarity values in the lists and returns the average value. The return similarity value is then assigned to its corresponding position in the similarity matrix, *line* 12. Before determining the similarity between another schema pair, the matching result list must be reinitialized, *line* 8.

Example 8.1 *Applying Algorithm 10 to XML schemas shown in Fig. 8.5, we get the following matrix*

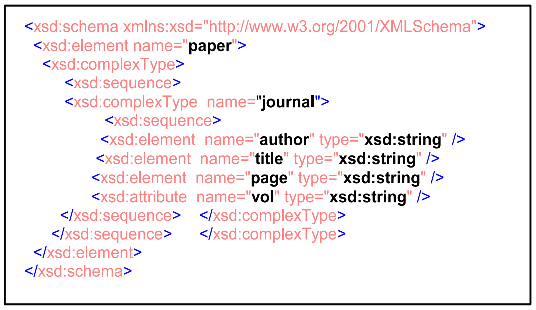
$$SSimMat = \begin{pmatrix} 1 & 0.478 & 0.41 \\ & 1 & 0.53 \\ & & 1 \end{pmatrix} \tag{8.2}$$

This simple example shows that the similarity value between $S1$ and $S2$ is less than that between $S2$ and $S3$. In contrast to the tree-editing approach, this gives the possibility that clustering this data set (may) results in $S2$ and $S3$ in a separate cluster than $S1$.

8.2.2 Clustering algorithms

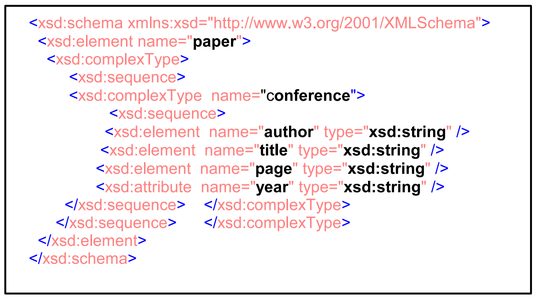
There are many techniques for clustering algorithms. Among them are hierarchical clustering algorithms [82]. Hierarchical clustering solutions are in the form of trees called *dendrograms*, which provide a view of the data at different levels of abstraction. The consistency of clustering solutions at different levels of granularity allows flat partitions of different granularity to be extracted during the data analysis, making them ideal for interactive exploration and visualization [80, 150]. There are two primary methods to obtain hierarchical clustering solutions: *agglomerative algorithms* and *partitional algorithms*.

In agglomerative algorithms, objects are initially assigned to its own cluster and then the pairs of clusters are repeatedly merged until the whole tree is formed. Thus, these algorithms build the dendrograms from bottom up. The key parameter in agglomerative



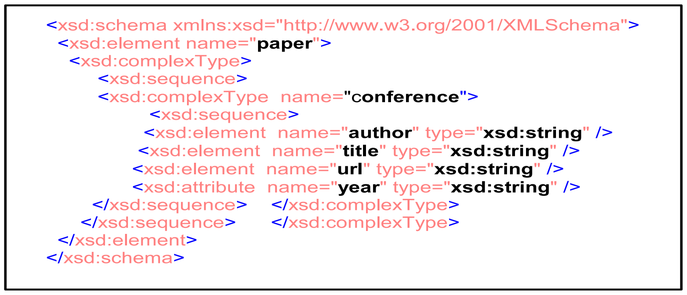
```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="paper">
  <xsd:complexType>
    <xsd:sequence>
    <xsd:complexType  name="journal">
        <xsd:sequence>
       <xsd:element  name="author" type="xsd:string" />
      <xsd:element  name="title" type="xsd:string" />
     <xsd:element  name="page" type="xsd:string" />
     <xsd:attribute  name="vol" type="xsd:string" />
   </xsd:sequence>   </xsd:complexType>
  </xsd:sequence>    </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

XML schema $S1$



```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="paper">
  <xsd:complexType>
    <xsd:sequence>
    <xsd:complexType  name="conference">
        <xsd:sequence>
       <xsd:element  name="author" type="xsd:string" />
      <xsd:element  name="title" type="xsd:string" />
     <xsd:element  name="page" type="xsd:string" />
     <xsd:attribute  name="year" type="xsd:string" />
   </xsd:sequence>   </xsd:complexType>
  </xsd:sequence>    </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

XML schema $S2$



```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="paper">
  <xsd:complexType>
    <xsd:sequence>
    <xsd:complexType  name="conference">
        <xsd:sequence>
       <xsd:element  name="author" type="xsd:string" />
      <xsd:element  name="title" type="xsd:string" />
     <xsd:element  name="url" type="xsd:string" />
     <xsd:attribute  name="year" type="xsd:string" />
   </xsd:sequence>   </xsd:complexType>
  </xsd:sequence>    </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

XML schema $S3$

Figure 8.5: XML schemas of XML data shown in Fig.8.1

algorithms is the method used to determine pair of clusters to be merged at each step. This task can be achieved by selecting the most similar pair of clusters. Three common methods have been proposed and used to realize this task:

- *Single-link*; This method measures the similarity between two clusters by the maximum similarity between XML data from each cluster. That is, the similarity between two clusters $Clust_J$ and $Clust_I$ is given by

$$Sim_{single}(Clust_I, Clust_J) = \max_{DT_i \in Clust_I, DT_j \in Clust_J} \{Sim(DT_i, DT_j)\} \tag{8.3}$$

- *Complete-link*; This method measures the similarity between two clusters by the minimum similarity between XML data from each cluster. That is,

$$Sim_{complete}(Clust_I, Clust_J) = \min_{DT_i \in Clust_I, DT_j \in Clust_J} \{Sim(DT_i, DT_j)\} \tag{8.4}$$

- *Group average (UPGMA)*; This method measures the similarity between two clusters as the average similarity between XML data from each cluster. That is,

$$Sim_{UPGMA}(Clust_I, Clust_J) = \frac{1}{n_i n_j} \sum_{DT_i \in Clust_I, DT_j \in Clust_J} \{Sim(DT_i, DT_j)\} \qquad \boxed{8.5}$$

Agglomerative algorithms have the feature that it is easy for them to group XML data that form small and cohesive clusters. However, they are computationally expensive and they have the disadvantage that if XML data are not part of cohesive groups, the initial merging decision may contain some errors, which tend to be multiplied during the clustering process [80, 150].

Unlike agglomerative algorithms, partitional clustering algorithms build the hierarchical solution from top to down by using a repeated bisectional approach [149]. In this approach, all data sets are initially partitioned into two clusters. Each cluster containing more than one XML data is selected and bisected. The process of bisection continues till each cluster contains one XML data. A key parameter of these algorithms is the use of a global criterion function whose optimization drives the clustering process. There are serval clustering criterion functions that optimize many aspects of intra-cluster similarity, inter-cluster dissimilarity, and their combinations [149, 150].

Partitional algorithms present different advantages. One of these advantages is that these algorithms have low computational requirement. Another advantage of these algorithms is that they use information about the entire collection of the data sets when they partition the data sets into a number of clusters. Thus, partitional algorithms are well-suited for clustering large datasets due to their relatively low computational requirements. However, the agglomerative algorithms outperform partitional algorithms. For this, in our implementation we make use of another hierarchical clustering algorithm called the *constrained agglomerative algorithm* [80].

Constrained agglomerative clustering algorithm

To gain features introduced by both agglomerative and partitional algorithms, the constrained agglomerative clustering algorithm has been proposed. The advantage of this algorithm is that it is able to benefit from the global view of collection used by the partitional algorithms and the local view of agglomerative algorithms. Moreover, the computational complexity of this algorithm will be improved over that of agglomerative algorithms. This gain can be achieved by using a partitional clustering algorithm to constrain the space over which the agglomerative clustering algorithm is performed by only allowing each XML data tree to merge with other XML data that are from the same partitionally discovered cluster.

The following are the steps needed to perform a constrained agglomerative clustering algorithm [150]:

- A partitional clustering algorithm is used to produce a $k - way$ clustering solution. These clusters are known as *constrained clusters*.
- Each constrained cluster is treated as a separate data set, and an agglomerative clustering algorithm is used to construct a dendrogram for each one.
- The k dendrograms are combined into a single one by merging them using an agglomerative algorithm.

8.3 Experimental evaluation

In this section we describe the experiments that we have carried out to evaluate our proposed framework. In the rest of this section we first describe the used datasets and our experimental methodology, followed by a description of the experimental results.

8.3.1 Data set

We used two different data sets depending on the evaluation criteria, as shown in Table 8.1. Part (A) is used to validate the clustering solution quality, while Part (B) is used to evaluate the clustering solution efficiency. These data sets have been obtained from different domains [2] [3] [4] and represent different characteristics. Each domain consists of a number of different categories that have structural and semantic differences. XML schemas from the same domain also vary in structures and semantics.

8.3.2 Experimental methodology and metrics

Different data sets are first extracted and modified to be ready for the clustering framework. The current implementation supports only clustering XSD schemas, hence we transformed DTDs into XSDs. The quality of clustering solutions has been verified using

[2]http://www.dbis.informatik.uni-goettingen.de/Mondial/
[3]http://www.cs.washington.edu/research/xmldatasets/
[4]http://www.cs.toronto.edu/db/clio/testSchemas.html

Table 8.1: Data set details

Part (A): Quality				*Part (B): Efficiency*		
Domain	No. of schemas	No. of nodes	No. of levels	Domain	No. of schemas	size
Auction	4	35/39	4/6	Genex	2	18KB
Mondial	7	11-	4/8	Auction	4	12KB
Financial	2	14/14	3/6	Book	15	23KB
TPC-H	10	8/45	2/6	ACM SIGMOD	12	76KB
GeneX	2	75/85	3/8	University	90	980KB
University	25	8-20	3/7	XCBL	90	590KB
				OAGIS	13	1.2MB

two common measures: (1) FScore as an external measure, and (2) intra-clustering similarity and inter-clustering similarity as internal measures, while the response time is used as a measure for the efficiency of the proposed approach.

FScore[5] is a trade-off between two popular information retrieval metrics, precision P and recall R. Precision considers the rate of correct matches in the generated solution, and recall considers the rate of correct matches in the model solution. Given a cluster C_i, let TP be the number of XML data in C_i which are similar (correctly clustered), FP be the number of documents C_i which are not similar (misclustered), FN be the number of documents which are not in C_i but should be. Precision and recall of a cluster C_i are defined as $P_i = \frac{TP}{TP+FP}$, and $R_i = \frac{TP}{TP+FN}$.

FScore combining precision and recall with equal weights for the given cluster C_i is defined by, $FScore_i = 2 \times \frac{P_i \times R_i}{P_i + R_i}$. The FScore of the overall clustering approach is defined as the sum of the individual class FScores weighted differently according to the number of XML data in the class

$$FScore = \frac{\sum_{i=1}^{k} n_i \times FScore_i}{n} \tag{8.6}$$

where k, n_i and n are the number of clusters, the number of XML data in a cluster C_i, and the number of XML data, respectively. A good clustering solution has the FScore value closer to one.

The internal clustering solution quality measures are evaluated by calculating the average inter and intra-clustering similarity. The intra-clustering similarity measures the cohesion within a cluster, how similar the XML data within a cluster are. This is computed by measuring the similarity between each pair of data within a cluster, and the intra-

[5]We use this label here which is more common for the data clustering community, it is named F-measure elsewhere in the thesis.

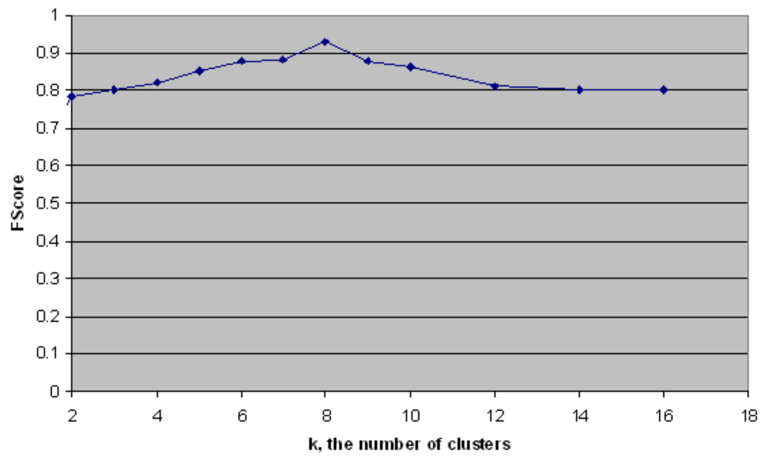


Figure 8.6: FScore

clustering similarity of a clustering solution is determined by averaging all computed similarities taking into account the number of XML data within each cluster

$$IntraSim = \frac{\sum_{i=1}^{k} n_i \times IntraSim(C_i)}{n}. \tag{8.7}$$

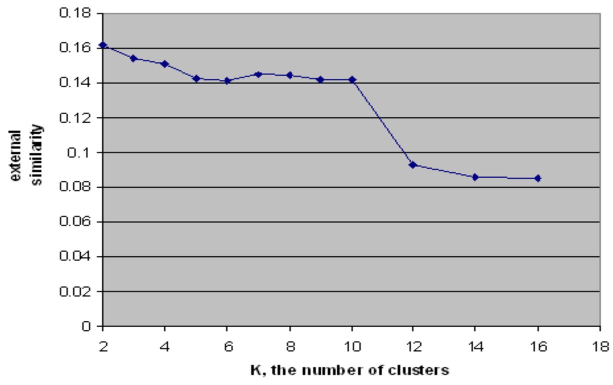
The larger the values of the intra-clustering similarity ($IntraSim$) are, the better is the clustering solution. The inter-clustering similarity measures the separation among different clusters. It is computed by measuring the similarity between two clusters. A good clustering solution has lower inter-clustering similarity values.

The response time, a measure for the framework efficiency, is the time required for the clustering framework to perform its task. This includes all specified three phases in the framework.

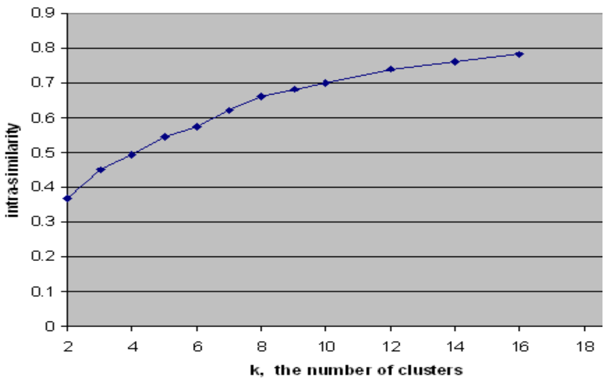
8.3.3 Experimental results

Quality evaluation

Data set Part(A) illustrated in Table 8.1 is used through quality evaluation utilizing both internal and external measures. Figure 8.6 illustrates the FScore of the data sets over 16 different clustering solutions. With $k = 2$, all the 25 schemas from the university domain are in one group, while the other schemas from the other domains are in the second group. This results in a high FScore at $k = 2$. As k increase, FScore increases until the best FScore occurs at $k = 8$. When the process reaches the 12 clustering solutions,



(a) Inter-clustering similarity



(b) Intra-clustering similarity

Figure 8.7: Internal quality measures

the clustering quality is stabilized. Fig. 8.6 also shows that the quality (FScore) of our proposed algorithm ranges between 79% and 93%, i. e. it is almost accurate.

The better clustering solution is the one having both higher intra-clustering similarity and lower inter-clustering similarity. Figure 8.7 supports this fact. The figure shows that as the clustering process continues, clusters are further decomposed into smaller sub-clusters that contain highly similar schemas, and the similarity between XML data within a cluster (intra-similarity) increases, as shown in Fig. 8.7. The figure also illustrates that as the number of clusters increases, the similarity between individual clusters (inter-similarity) decreases.

Scalability evaluation

In this case, we carried out two sets of experiments. The first involves the whole data set to observe the effect of large-scale schema (OAGIA data set), while the second does not involve the OAGIA data set. The results are reported in Fig. 8.8, which shows that our approach scales well especially in the second experiment. The figure also illustrates that our system could group similar XML schemas across 226 schemas with a size of 3MB in a time of 900 seconds, while the approach needs 150 seconds to cluster 213 schemas.

8.4 Related Work

The relationship between XML schema clustering and schema matching is bidirectional. From the viewpoint of using clustering to support and improve schema matching, research

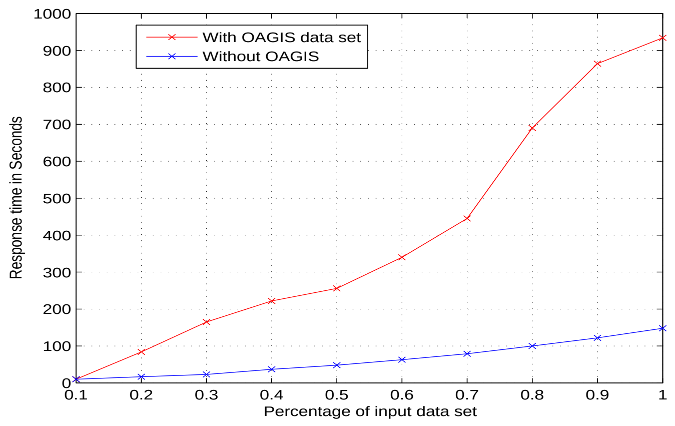


Figure 8.8: The framework response time.

in this direction depends heavily on the fact that it is easier to find element correspondences between schemas that are contextually similar. The approach proposed in [112] develops a clustered-based approach to schema matching. The approach clusters schemas based on their contextual similarity, and then it clusters attributes of schemas within the same schema cluster. Afterwards, attributes across different schema clusters are clustered using statistical information gleaned from existing attribute clusters to find attribute correspondences among different schemas. However, the approach deals only with flat schemas. Authors in [129] propose a clustered schema matching technique. Clustering is used to identify clusters in the large schema repository which are likely to produce mappings for a personal schema. Other approaches, which make use of clustering to identify element correspondences in the context of integrating heterogeneous data sources, can be found in [148, 113].

From the other point of view, research on clustering XML data is gaining momentum. Based on the data to be clustered, XML data clustering can be broadly classified into two categories: clustering XML documents and clustering XML schemas. Many approaches have been developed in the context of XML document clustering [73], while only little work is done in the context of XML schema clustering [89, 108]. [89] proposed an integration strategy, called XClust, that involves the clustering of DTDs. A matching algorithm, based on the semantic and structural properties of schema' elements has been proposed. [108] also developed a framework, called XMine, to cluster XML schemas (both DTTs and XSDs). XMine makes use of semantic, syntactic and structural properties of schema' elements.

Both XClust and XMine, as our proposed framework, represent XML schemas as rooted (ordered) labeled trees. However, we extend the tree representation of XML schemas into a sequence representation in order to efficiently deal with schema elements instead of traversing schema trees many times. Moreover, the two clustering frameworks make use of WordNet to determine semantic (synonyms) similarity. XMine additionally implements a user-defined dictionary in order to identify abbreviations and finally makes use of syntactic string functions (string edit distance) to compare between element names if no semantic relationships exist. In contrast, we only use simple string functions in order to determine initial similarity values for the structural matcher. Our structural matcher is similar to the one in [89]. They both depend on the node context utilizing both ancestor and descendant contexts of a node. However, our approach benefits from the sequence representation of schema trees.

8.5 Summary

In this chapter, we introduced a schema matching-based approach to cluster XML schemas showing the deployment of our schema matching algorithms in several application domain. In particular, we developed and implemented a clustering framework that consists of three phases: Pre-processing; to represent XML schemas as sequence representations, Similarity computation; to determine the similarity across XML schemas, and Clustering; to group similar XML schemas into clusters using the hierarchical clustering algorithm. To validate the performance of the proposed framework, we conducted a set of experimental evaluation. Through our validation, we measured both the clustering solution quality and the clustering scalability. The experimental evaluation showed that our proposed framework is almost accurate with FScore ranging between 80% and 93%, and the evaluation showed that the framework scaled well w.r.t. large number and large-scale schemas.

9

SeqDisc: Supporting Web service discovery

In the previous chapter we showed that how we could effectively and efficiently deploy our schema matching approach in clustering XML data. In this chapter, we strive the deployment of the approach in another application domain, *Web service discovery*.

Locating desired Web services has become a challenging research problem due to the vast number of available Web services within an organization and on the Web. This necessitates the need for developing flexible, effective, and efficient Web service discovery frameworks. To this purpose, both the semantic description and the structure information of Web services should be exploited in an efficient manner. In this chapter, we aim to put our developed schema matching techniques into use within the context of Web service discovery. In particular, we present a flexible and efficient service discovery approach, called *SeqDisc*, which is based on the use of the Prüfer encoding method to construct a one-to-one correspondence between Web services and sequence representations. First, we motivate the Web service discovery process. We then introduce basic concepts related to Web services. We finally describe and experimentally evaluate our Web service discovery approach.

The material presented in this chapter has been developed and published in [16, 6].

9.1 Introduction

The continuous and rapid evolution of service-oriented technologies provides methods for systems development and integration where systems group functionality around business processes and package these as interoperable services. As a result, Web services have emerged as one of the distributed computing technologies, and sparked a new round of interest from research and industrial communities. Web, once solely a repository for text and images, is evolving into a *provider of services*, such as flight information providers, temperature sensors, and *world-altering services*, such as flight booking programs, and a variety of e-commerce and business-to-business applications [99]. These web-based applications rely heavily on permitting generic software components to be developed and shared. With the adoption of XML as a standard and common language for information representation and exchange on the Web, the underlying principles have been gained wide scale adoption through the definition of Web service standards. Web services are well-defined, reusable software components that perform specific, encapsulated tasks via standardized Web-oriented mechanisms [38]. They can be discovered, invoked, and the composition of several services can be choreographed using well-defined workflow modeling frameworks. Based on this, the research community has identified two major areas of interest: *Web service discovery* and *Web service composition* [95]. In this chapter, we present the issue of locating Web services efficiently.

As the number of Web services increases, the problem of locating Web services of interest from a large pool becomes a challenging research problem [140, 56, 76, 36]. Several solutions have proposed, however, most of them suffer from the following disadvantages:

- A large number of these solutions are syntactic-based. Several *simple search engines*, which provide only *simple keyword search* on Web service descriptions, and *traditional attribute-based matchmaking algorithms* have been proposed. In the Web service discovery context, it becomes apparent that keyword search and attribute-based mechanisms are insufficient since they do not capture the underlying semantic of Web services and/or they partially satisfy the need of user search. This is due to the fact that keywords are often described by a natural language. As a result, the number of retrieved services with respect to the keywords are huge and/or the retrieved services might be irrelevant to the need of their consumers [95]. More recently, this issue sparked a new research into the Semantic Web where some research uses ontology to annotate the elements in Web services [19, 109]. Nevertheless, integrating different ontologies may be difficult while the creation and maintenance of ontologies may involve a huge amount of human effort. To address the second aspect, clustering algorithms are used for discovering Web services. However, they are based on keyword search [56, 109, 95].

- Most of the existing approaches are not scale well. This means that they are not able to scale to large-scale and to large numbers of services, service publishers, and service requesters. This is due to the fact that they mostly follow a centralized registry approach. In such an approach, there is a registry that works as a store of WS advertisements and as the location where service publication and discovery takes place. The scalability issue of centralized approaches is usually addressed with the help of replication (e.g., UDDI). However, replicated Registries have high operational and maintenance cost. Furthermore, they are not transparent due to the fact that updates occur only periodically

We see Web service discovery as a matching process, where available services' capabilities satisfy a service requester's requirement. Hence the problem of locating the desired service is converted to a matching process. As we stated, two main aspects should be considered during solving the matching process: the *quality* of the discovered service and the *efficiency* especially in large-scale environments. To obtain a better quality, not only is the textual description of Web services sufficient, but also the underlying structures and semantics should be exploited. Also to get a better performance, an efficient methodology should be advised.

To this context, in this chapter, we propose a flexible and efficient approach, called *SeqDisc*, for assessing the similarity of Web services, which can be used to support locating Web services on the Web. We first represent Web service document specifications described in WSDL as rooted, labeled trees, called *service trees*. By investigating service trees, we observe that each tree can be divided into two parts (subtrees), namely the *concrete* and *abstract* parts. We discover that the concrete parts from different WSDL documents have the same hierarchal structure, but may have different names. Therefore, we develop a *level matching* approach, which computes the name similarity between concrete elements at the same level. However, the abstract parts of the WSDL documents have differences in structure and semantics. To efficiently access the abstract elements, we represent them using the Prüfer encoding method [115], and then apply our sequence-based schema matching approach to the sequence representation. A set of experiments is conducted in order to validate our proposed approach employing real data sets. The experimental results showed that the approach is accurate and scale well.

The remainder of the chapter is organized as follows: We briefly discuss some of the research work related to the Web service discovery problem. We then provide basic concepts and definitions used throughout the chapter. An overview of the SeqDisc approach is discussed. Different methods advised to assess the similarity between Web services are then described. We finally report on experimental results.

9.2 Related Work

As pointed out by [69], web service discovery mechanisms originated from the agent match-making scheme. It consists of a set of general activities, including advertising and storing web services to middle agents, asking a middle agent for a specific request, and answering the user request. Furthermore, the discovery mechanisms differ according to the way the web services are specified. In general, two common languages have been widely used: WSDL[1] is popular and adopted by the industry due to its simplicity, while OWL-S[2] and WSMO[3] are well accepted by researchers as they offer much structured and detailed semantic mark-ups. The discussion presented in section is based on the way the services themselves are modeled, i.e. the representation of Web services. In general there are two techniques to represent Web services: the *Information Retrieval (IR) approach* and the *semantics approach* [69].

9.2.1 Information Retrieval Approach

Keyword-based is one possible and the simplest approach to do web service discovery. However, it suffers from several shortcomings, such as it does not exploit well-defined semantics and is limited because of the ambiguity of natural language. To tackle the inadequacy of keyword-based Web service discovery an approach was proposed in [121]. The key concept in the approach is to represent service description as document vectors, a dominant approach in the IR field [23]. A description text corresponds to a vector V in the vector space spanned by all terms used in the service description texts. Authors go one step further by representing all document vectors as columns in a term-document matrix A.

Other approaches, which combine IR techniques [23] with structure and/or schema matching have been developed, e.g. [140, 76, 132, 85]. The approach, in [140], is based on information retrieval and structure matching. Given a potential partial specification of the desired service, all textual elements of the specification are extracted and are compared against the textual elements of the available services, to identify the most similar service

[1]http://www.w3.org/TR/wsdl20/
[2]http://www.daml.org/services/owl-s/
[3]http://www.wsmo.org/wsml/

description files and to order them according to their similarity. Next, given this set of likely candidates, a structure-matching method further refines the candidate set and assesses its quality. The drawback is that simple structural matching may be invalid when two web-service operations have many similar substructures on data types. In [76], authors develop an approach which uses the traditional IR technique TF (term frequency) and IDF (inverse document frequency). They extract words from Web service operation descriptions in WSDL. These words are pre-processed and assigned weights based on IDF. According to these weights, the similarity between the given description and a Web service operation description can be measured. After obtaining candidate operations, a schema-based method to measure similarity among them using tree edit distance is applied. However, using the tree edit distance method is a computational cost process, which declines the Web service discovery performance.

The author in [85] recognizes the fact that keyword-based matching methods can help users to locate quickly the set of useful services, but they are insufficient for automatic retrieval, and the high cost of formal ontology-based methods alienates service designers from their use in practice. Therefore, to assess the similarity between Web services, the author makes use of several IR techniques, such as the Vector Space model and WordNet. Further, the proposed approach combines the lexical and structural information of several concepts of Web service description to assess the similarity. The approach, in [132], developed a set of methods that assess the similarity between two WSDL specifications based on the structure of their data types and operations and the semantics of their natural language descriptions and identifiers. Given only a textual description of the desired service, a semantic information-retrieval method can be used to identify and order the most relevant WSDL specifications based on the similarity of the element descriptions of the available specifications with the query. If a (potentially partial) specification of the desired service behavior is also available, this set of likely candidates can be further refined by a semantic structure-matching step, assessing the structural similarity of the desired vs the retrieved services and the semantic similarity of their identifiers. Our approach is similar to [76], but we focus on both structure and semantic matching like [132, 85]. However, we exploit more information about Web service and advice a level and schema matching algorithm. Further, to improve the performance of the discovery process, we represent Web services as sequences using the Prüfer encoding method [115].

9.2.2 Semantics Approach

Recent prototypes have been proposed to improve the accuracy of Web service discovery by applying various ontology based discovery frameworks [79, 4, 33] and data mining techniques [109, 107, 36]. The framework in [79] uses ontologies to discover the Web services that best match a specific operation domain. The available data are represented with domain ontologies, and the available operations with operation ontologies. Generalization relationships on both models are encoded in XML formats. In [33], the authors propose an ontology-based hybrid approach where different kinds of matchmaking strategies are combined together to provide a flexible service discovery environment. The approach extends the keyword-based method based on the use of two ontologies: *domain* and *service*. It considers three different matchmaking models: (1) a deductive model to determine the kind of match; (2) a similarity-based model to measure the degree of match between services and (3) a hybrid model to combine the previous ones. In [109], authors concentrate on Web service discovery with OWL-S and clustering technology, which consists of three main steps. The OWL-S is first combined with WSDL to represent service semantics before a clustering algorithm is used to group the collections of heterogeneous services together. Finally, a user query is matched against the clusters, in order to return the suitable services.

9.3 Preliminaries

In this section we present some definitions and basic preliminaries concerning Web services and Web service modeling.

9.3.1 Web services

A Web service is a software component identified by an URI, which can be accessed via the Internet through its exposed interface. The interface description declares the operations which can be performed by the service, the types of messages being exchanged, and the physical location of ports, where information should be exchanged.

Web service architecture

Three fundamental layers are required to provide or use Web services [22]. First, Web services must be network-accessible to be invoked, HTTP is the de-facto standard network

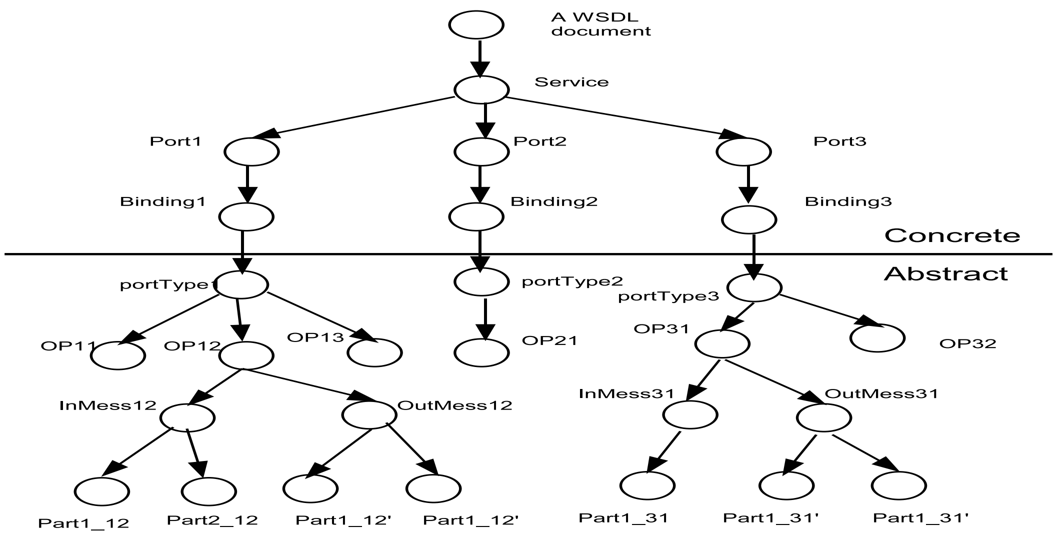


Figure 9.1: A hierarchal representation of a WSDL document.

protocol for Internet available Web services. Second, Web services should be XML-based messaging for exchanging information, and SOAP[4] is the chosen protocol. Finally, it is through a service description that all the specification for invoking a Web service are made available; WSDL[5] is the de-facto standard for XML-based service description.

Web service description

The Web Service Description Language (WSDL) is an XML-based language to describe Web services and how to access them. A Web service is seen as a set of end points operating on messages containing either document-oriented or procedure-oriented data. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete set of operations are bundled into abstract endpoints (services). Typically, a WSDL document specification contains the following elements, as shown in Fig. 9.1:

- *types*; The *types* element encloses data type definitions that are relevant for the exchanged messages. For maximum interoperability and platform neutrality, WSDL prefers the use of XSD (XML schema) as the canonical type system.
- *message*; A *message* element represents an abstract, typed definition of the data being communicated. A message consists of one or more logical parts, each of which

[4]http://www.w3.org/TR/soap/
[5]http://www.w3.org/TR/wsdl20/

is associated with a type system using a message-typing attribute.

- *operation*; An *operation* element is an abstract description of an action supported by the service.
- *portType*; The *portType* element is an abstract set of operations. Each operation refers to an input message and output messages.
- *binding*; A *binding* element specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
- *port*; A *port* element is a single endpoint defined as a combination of a binding and a network address.
- *service*; The *service* element is a collection of related endpoints (ports).

A WSDL description is typically composed of an abstract part and a concrete part. The abstract part of a service definition contains WSDL elements that can be reused and referenced by multiple service implementations, such as *binding, portType, message* and *types* elements. The concrete part contains WSDL elements that describe how a particular service is implemented, such as *service* and *port* elements. Figure 9.1 depicts the concepts defined by a WSDL document.

9.3.2 Web service representation

In order to capture both semantic information and structural information of a WSDL document, we model it as a rooted labeled tree. Following Definition 5.1, a WSDL document is represented as rooted labeled tree, called service tree ST, $ST = (N, E, Lab)$, where N is the set of nodes representing WSDL document elements, E is the set of edges representing the parent-child relationship between WSDL document elements, and Lab is a set of labels associated to WSDL document elements describing the properties of them. All WSDL document elements, except the *part* elements, have two main properties: the *type* property to indicate the type of the element (*port, binding, operation,...*), and the *name* property to distinguish between similar type elements.

Figure 9.1 also delineates the hierarchal structure of a WSDL document. The figure indicates that a *service* consists of a set of *ports*, each contains only one *binding*. A binding contains only one *portType*. Each portType consists of a set of *operations*, each contains an *input message* and *output messages*. A message consists of a set of *parts*, where each part describes the logical content of the message.

From the hierarchal structure of a Web service tree, we divide its elements into a *concrete* part and *abstract* part. The intuition for this classification is that service trees representing different web services have the same structure from the root node to the part node, while the structure of the remaining depends on the content of operation messages. The following are definitions for concrete and abstract parts of a service tree.

Definition 9.1 *A concrete part of a service tree (ST) is the subtree (ST_C) extending from the root node to the portType element, such that $ST_C = \{N_C, E_C, Lab_C\} \subset ST$, $N_C = \{n_{root}, n_{port1}, n_{binding1}, n_{portType1}, ..., n_{portType_l}\} \subset N$, where l is the number of concrete elements in the service tree.*

Definition 9.2 *An Abstract part of a service tree (ST) is the set of subtrees rooted at operation elements, such that $ST_A = \{ST_{A_1}, ST_{A_2}, ..., ST_{A_k}\}$, where k is the number of operations in the service tree.*

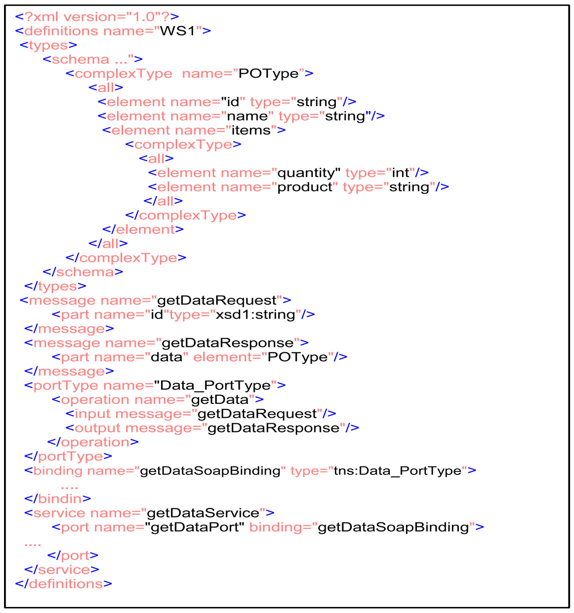
This means that a service tree comprises a concrete part and an abstract part, i.e., $ST = ST_C \cup ST_A$. To assess the similarity between two web services, we consequently compare their concrete and abstract parts. Thus the problem of measuring similarity between Web services is converted into the problem of tree matching.

9.3.3 An illustrative example

Let us now introduce an illustrative example of assessing the similarity between two Web services, which is taken from [140]. As shown in Fig. 9.2, we have two Web services described by two WSDL documents $WS1$ and $WS2$, respectively. $WS1$ contains one operation , *getData*, that takes a string as input and returns a complex data type named *POType*, which is a product order. The second WSDL document contains one operation, *getProduct*, that takes an integer as input and returns the complex data type *MyProduct* as output.

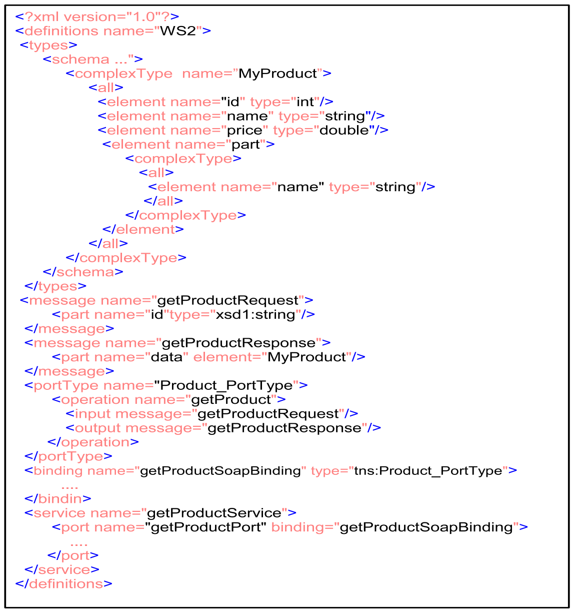
9.4 SeqDisc: the discovery framework

Our proposed approach is based on the exploitation of the structure and semantic information from WSDL documents. The objective is to develop an efficient approach that



```
<?xml version="1.0"?>
<definitions name="WS1">
 <types>
    <schema ...">
        <complexType  name="POType">
            <all>
             <element name="id" type="string"/>
             <element name="name" type="string"/>
              <element name="items">
                  <complexType>
                    <all>
                     <element name="quantity" type="int"/>
                     <element name="product" type="string"/>
                    </all>
                  </complexType>
              </element>
            </all>
        </complexType>
    </schema>
  </types>
 <message name="getDataRequest">
      <part name="id"type="xsd1:string"/>
  </message>
  <message name="getDataResponse">
      <part name="data" element="POType"/>
  </message>
  <portType name="Data_PortType">
      <operation name="getData">
        <input message="getDataRequest"/>
        <output message="getDataResponse"/>
     </operation>
  </portType>
  <binding name="getDataSoapBinding" type="tns:Data_PortType">
        ....
  </bindin>
  <service name="getDataService">
      <port name="getDataPort" binding="getDataSoapBinding">
  ....
     </port>
  </service>
</definitions>
```

(*a*) WS1: getData Web service.



```
<?xml version="1.0"?>
<definitions name="WS2">
 <types>
    <schema ...">
        <complexType  name="MyProduct">
            <all>
             <element name="id" type="int"/>
             <element name="name" type="string"/>
             <element name="price" type="double"/>
              <element name="part">
                  <complexType>
                    <all>
                     <element name="name" type="string"/>
                    </all>
                  </complexType>
              </element>
            </all>
        </complexType>
    </schema>
  </types>
 <message name="getProductRequest">
      <part name="id"type="xsd1:string"/>
  </message>
  <message name="getProductResponse">
      <part name="data" element="MyProduct"/>
  </message>
  <portType name="Product_PortType">
      <operation name="getProduct">
        <input message="getProductRequest"/>
        <output message="getProductResponse"/>
     </operation>
  </portType>
  <binding name="getProductSoapBinding" type="tns:Product_PortType">
        ....
  </bindin>
  <service name="getProductService">
      <port name="getProductPort" binding="getProductSoapBinding">
         ....
     </port>
  </service>
</definitions>
```

(*b*) WS2: getProduct Web service.

Figure 9.2: Two Web service specifications.

measures the similarity between Web services. The measured similarity is then used as a guide to locate the desired Web service.

The basic idea underlying our research is that, although WSDL does not explicitly provide support for semantic specifications, it does contain information that can potentially be used to infer the semantics of the specified Web service [132]. (1) The WSDL specification contains several elements whose features are textual descriptions, explaining types, parts, and operations of the service. (2) The internal structure of operation types, represented in the XML format, is designed to capture the domain-specific relations of various data required and provided by the service. (3) Each WSDL element has a name feature that is also usually meaningful.

To realize this goal, we first analyze WSDL documents and represent them as service trees using Java APIs for WSDL (JWSDL) and a SAX parser for the contents of the XML schema (the types element). Then, each service tree is examined to extract its concrete and its abstract parts. We subsequently develop a level matching algorithm to measure the similarity between concrete parts from different service trees, while to assess the similarity between abstract parts, we propose a sequence-based matching algorithm. By this mechanism we gain a high flexibility in determining the similarity between web services. As it will be shown in the experimental evaluation, we have two possibilities

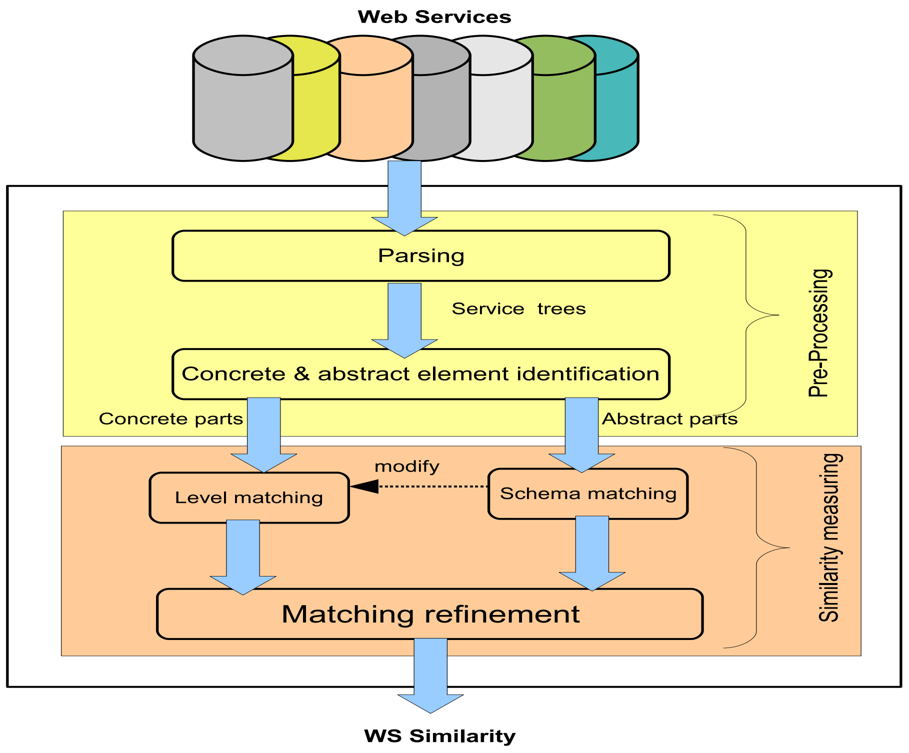


Figure 9.3: Web services similarity measure framework.

to compute the similarity. The first is to only exploit abstract parts (operations), while the second is to use both the abstract and concrete parts. Furthermore, the proposed approach scales well. As it will be shown, the level matching algorithm has a linear time complexity as a function of the number of concrete elements, while the sequence-based matching algorithm benefits from the sequence representation to reduce the time complexity. Figure 9.3 illustrates the outline of the proposed approach.

The outline of the algorithm implementing the proposed framework is shown in Algorithm 11. The algorithm accepts a set of Web service specifications as input, $WS_1, WS_2, ..., WS_n$, to assess the similarity between them as output. The algorithm constructs a web service similarity matrix, $WSSimMat$. To this, we first analyze each WSDL document and represent it as a service tree, *line* 3, using Java APIs for WSDL(JWSDL)[6]and a SAX parser[7] for the contents of the XML schema (the *types* element). Then, each service tree is examined to identify and extract the concrete part, *line* 4, and a set of abstract parts, *line* 5, of the service tree. In fact, all these tasks, rep-

[6]http://www.jcp.org/en/jsr/detail?id=110
[7]http://www.saxproject.org

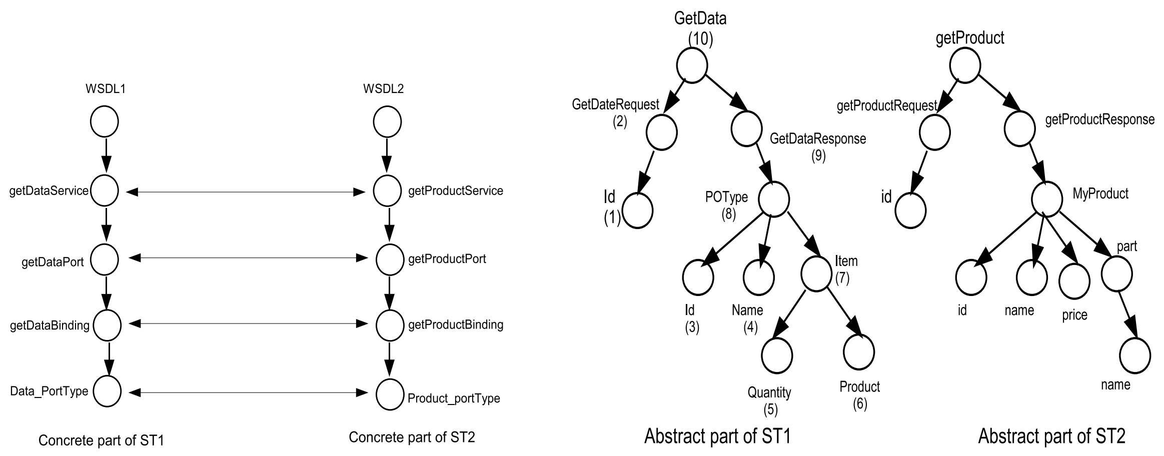
resenting web services as service trees and identifying both abstract and concrete parts, are performed at the same time to improve the performance. This means that during the parsing phase of each WSDL document, we identify and construct concrete and abstract parts of its service tree. We observe that concrete parts from different service trees have the same hierarchical structure, therefore we develop a level matching method to measure the similarity between their elements, *line* 13. To measure the similarity between abstract parts, we use our developed schema matching algorithm, *line* 14. Finally, the output of the level match and schema matching algorithms are refined in a refine match algorithm, *line* 15, to determine the total similarity between every Web service pair, and store it in the corresponding position in the similarity matrix. In the following section, we describe in details the similarity measuring phase.

Algorithm 11: Web service similarity algorithm

```
input : A set of web service specifications, $WS_1$, $WS_2$, ..., $WS_n$
output: A WS similarity matrix, $WSSimMat$
1  $WSSimMat[][] \leftarrow 0$;
2  for $i \leftarrow 1$ to $n$ do
3      $ST[i] \leftarrow$ buildST ($WS_i$);
4      $concrete[i] \leftarrow$ identifyConcrete ($ST[i]$);
5      $abstractList[i] \leftarrow$ identifyAbstract ($ST[i]$);
6  end
7  for $i \leftarrow 1$ to $n - 1$ do
8      $concrete1 \leftarrow (concrete$ [i]);
9      $abstractList1 \leftarrow (abstractList$ [j]);
10     for $j \leftarrow i + 1$ to $n$ do
11         $concrete2 \leftarrow (concrete$ [j]);
12         $abstractList2 \leftarrow (abstractList$ [j]);
13         $levSim \leftarrow$ levelMatch ($concrete1$, $concrete2$);
14         $schSim \leftarrow$ schemaMatch ($abstractList1$, $abstractList2$);
15         $WSSimMat[i][j] \leftarrow$refineMatch ($levSim$, $schSim$);
16     end
17 end
```

9.5 Similarity assessment

To assess the similarity between two web services given their WSDL specifications $WSDL1$ & $WSDL2$, we first represent them as service trees ($ST1$ & $ST2$). Each ser-



(a) Concrete parts of $ST1$ & $ST2$. (b) Abstract parts of $ST1$ & $ST2$.

Figure 9.4: Concrete & abstract parts of $ST1$ & $ST2$.

vice tree is then classified into concrete and abstract parts. The concrete parts from different service trees have the same hierarchal structure. Hence, the similarity between concrete parts of two web services is computed using only concrete part element names by comparing elements with the same level. We call this type of matching *level matching.* Abstract parts from different service trees have different structures based on the message contents. Therefore, we advise a *sequence-based schema matching* approach to measure the similarity between them.

During the discussion of similarity measuring algorithms, we refer to the two Web service documents $WS1$ & $WS2$ illustrated in Figure 9.2. Each WSDL document is parsed and represented as a service tree (ST), as shown in Fig. 9.4. The concrete and abstract parts of each service tree are then identified and extracted. Figure 9.4(a) presents the concrete parts of the two service trees, $ST1$ and $ST2$, while Fig. 9.4(b) gives abstract parts.

9.5.1 Level matching

Once obtaining the concrete parts of service trees, $ST_C1 \subset ST1$ and $ST_C2 \subset ST2$, we apply our level matching algorithm that linguistically compares nodes at the same level, as shown in Fig. 9.4(a). The level matching approach considers only semantic information of concrete elements. It measures the elements (tag names) similarity by comparing each pair of elements at the same level based on their names.

Algorithm 12 outlines the steps needed to determine level matching. The algorithm accepts the concrete parts of the service tress, ST_{C1}, ST_{C2}, and produces the name similarity between the elements of the concrete parts. It starts by initializing the matrices, wherein the name similarities are kept. We have three levels for each service tree, $line\,2$. When the loop index equals 1, $i = 1$, the algorithm deals with the port nodes, when $i = 2$ deals with the binding nodes, and with the portType nodes when $i = 3$. To compute the similarity between elements at the same level, the algorithm uses two inner loops, $lines\,3, \&, 5$. It first extracts the name of the node j at the level i, $line\,4$, and the name of the node k at the same level, $line\,6$. Then, the algorithm uses a name similarity function to compute the name similarity between the names of the nodes, $line\,7$. Finally, depending on the level, it stores the name similarity matrix into the corresponding element matrix.

Algorithm 12: Level matching algorithm

```
Require: Two concrete parts, $ST_{C1} \& ST_{C2}$
Ensure: $3 Name\ similarity\ matrices,\ NSimM$
 1: $PortSimM[][] \Leftarrow 0$, $BindSimM[][] \Leftarrow 0$, $PTypeSimM[][] \Leftarrow 0$;
 2: for $i = 1$ to 3 do
 3:    for $j = 1$ to $l$ do
 4:       $name_1 \Leftarrow getName(ST_{C1}(i, j))$;
 5:       for $k = 1$ to $l'$ do
 6:          $name_2 \Leftarrow getName(ST_{C2}(i, k))$;
 7:          $NSimM[i][j] \Leftarrow NSim(name_1, name_2)$;
 8:       end for
 9:    end for
10:    if $i = 1$ then
11:       $PortSimM \Leftarrow NSimM$;
12:    else if $i = 2$ then
13:       $BindSimM \Leftarrow NSimM$;
14:    else
15:       $PTypeSimM \Leftarrow NSimM$;
16:    end if
17: end for
```

To compute the name similarity between two element names represented as strings, we first break each string into a set of tokens T_1 and T_2 through a customizable tokenizer using punctuation, upper case, special symbols, and digits, e.g, getDataService $\rightarrow$ {get, Data, Service}. We then determine the name similarity between the two sets of name tokens T_1 and T_2 as the average best similarity of each token with a token in the other set. It is computed as follow:

$$Nsim(T_1, T_2) = \frac{\sum_{t_1 \in T_1}[\max_{t_2 \in T_2} sim(t_1, t_2)] + \sum_{t_2 \in T_2}[\max_{t_1 \in T_1} sim(t_1, t_2)]}{|T1| + |T2|} \qquad (9.1)$$

To measure the string similarity between a pair of tokens, $sim(t_1, t_2)$, we use two string similarity measures, namely the edit distance and trigrams [41]. The name similarity between two nodes is computed as the combination (weighted sum) of the two similarity values. The output of this stage is 3 $(l \times l')$ name similarity matrices, $NSimM$, where l is the number of concrete part elements of ST_{C1} and l' is the number of concrete part elements of ST_{C2} per level (knowing that the number of *ports*, the number of *bindings*, and the number of *protType* are equal). In the running example, see Fig. 9.4(a), $l = 1$ and $l' = 1$.

Algorithm Complexity The algorithm runs three times, one for every level. Through each run, it compares l elements of ST_{C1} with l' elements of the second concrete part. This leads to a time complexity of $O(l \times l')$, knowing that the number of elements in each level is very small.

Example 9.1 *Applying the level matching approach on the concrete parts shown in Fig. 9.4(a), we obtain the following result shown in Table 9.1 (notice that $n' = m' = 1$ in this example).*

Table 9.1: Example 9.1 result

$ST1$	$ST2$	$NSimM$
getDataPort	getProductPort	0.438
getDataBinding	getProductBinding	0.6607
Data_PortType	Product_PortType	0.545

9.5.2 Schema matching

In contrast to concrete parts, the abstract parts from different service trees have different structures. Therefore, to compute the similarity between them, we should capture both semantic and structural information of the abstract parts of the service trees. To realize this goal, we deploy our sequence-based matching approach developed in the previous chapters (Part III). As mentioned there, the approach consists of two stages: *Prüfer Sequence Construction* and *Similarity computation.* The Pre-processing phase is considered with the representation of each abstract item (subtree) as a sequence representation using the Prüfer encoding method. The similarity computation phase aims to assess the similarity between abstract parts of different service trees exploiting semantic and structural information to construct an operation similarity matrix.

The outline of the algorithm implementing the proposed schema matching approach is shown in *Algorithm* 13. The algorithm accepts two sets of abstract parts of the service trees input, $ST_{A1} = \{ST_{A11}, ST_{A12}, ..., ST_{A1k}\}$ and $ST_{A2} = \{ST_{A21}, ST_{A22}, ..., ST_{A2k'}\}$, to compute the similarity between them, where each item in the sets represents an operation in the service tree, k and k' are the number of operations in the two abstract parts, respectively. To this, we first analyze each operation (abstract item) and represent it as a sequence representation using the Prüfer encoding method, CPS, *line* 3& 6. Then, the algorithm proceeds to compare all CPS pairs to assess the similarity between every operation pair using our developed sequence matching algorithms, *line* 10. The returned similarity value is stored in its corresponding position in the operation similarity matrix, $OpSimM$. Finally, the similarity matrix is returned for further investigation, *line* 13.

Algorithm 13: Schema matching algorithm

```
Require: Two abstract parts, $ST_{A1} \& ST_{A2}$
    $ST_{A1} = \{ST_{A11}, ST_{A12}, ..., ST_{A1k}\}$
    $ST_{A2} = \{ST_{A21}, ST_{A22}, ..., ST_{A2k'}\}$
Ensure: Operation similarity matrix, OpSimM
 1: $OpSimM[][] \Leftarrow 0;$
 2: for $i = 1$ to $k$ do
 3:     $CPS_1[i] \Leftarrow buildCPS(ST_{A1i})$
 4: end for
 5: for $j = 1$ to $k'$ do
 6:     $CPS_2[j] \Leftarrow buildCPS(ST_{A2j})$
 7: end for
 8: for $i = 1$ to $k$ do
 9:     for $j = 1$ to $k'$ do
10:         $OpSimM[i][j] \Leftarrow computeSim(CPS_1[i], CPS_2[j]);$
11:     end for
12: end for
13: return OpSimM;
```

In the following, we first present an example of applying the CPS construction method introduced in Chapter 5 to abstract parts of service trees, we then give a brief discussion of how to apply sequence-based stages to assess the similarity between abstract parts.

Example 9.2 *Consider the abstract parts of the two service trees* $ST1$ *&* $ST2$ *shown in Figure 9.4(b).* $CPS(ST1) = (NPS, LPS)$, *where* $NPS(ST1) = (2, 10, 8, 8, 7, 7, 8, 9, 10, -)$ *and* $LPS(ST1).name =$ *(id, getDataReequest, id, name, quantity, product, item, POType, getDataResponse, getData).*

Matching Algorithms

In this stage, we aim to assess the similarity between abstract parts of service trees (operations). This task can be stated as follows: Consider we have two Web service document specifications $WS1$ and $WS2$, each containing a set of operations. $OpSet1 = \{op_{11}, op_{12}, ..., op_{1k}\}$ represents the operation set belonging to $WS1$, while $OpSet2 = \{op_{21}, op_{22}, ..., op_{2k'}\}$ is the operation set of $WS2$. The task at hand is to construct $k \times k'$ operation similarity matrix, $OpSimM$. Each entry in the matrix, $OpSimM[i][j]$, represents the similarity between operation op_{1i} from the first set and operation op_{2j} from the second one. The proposed matching algorithm operates on the sequence representations of service tree operations and consists of three steps, as described in the previous chapters in more details. We give here a quick overview of the approach to present more explanation.

1. *Linguistic matcher.* First, we make use of our linguistic algorithm developed in Part III to compute a degree of linguistic similarity for elements of service tree operation pairs exploiting semantic information represented in LPSs. The output of this step are $k \times k'$ linguistic similarity matrices, $LSimM$, where k is the number of operations in $ST1$ and k' is the number of operations in $ST2$. Equation 9.2 gives the entries of a matrix, where $Nsim(T_i, T_j)$ is the name similarity measure, $Tsim$ is a similarity function to compute the type/data type similarity between nodes, and $combine_l$ is an aggregation function that combines the name and data type similarities.

$$LSimM[i, j] = combine_l(Nsim(T_i, T_j), Tsim(n_i, n_j)) \tag{9.2}$$

2. *Structural matcher.* Once a degree of linguistic similarity is computed, we make use of the *structural algorithm* developed in Part III to compute the structural similarity between abstract part elements. As shown, to measure the structural similarity between two nodes, we compute the similarity of their child, sibling, ancestor, and leaf contexts, utilizing the structural properties carried by sequence representations of service trees. The output of this phase are $k \times k'$ structural

similarity matrices, $SSimM$. Equation 9.3 gives entries of a matrix, where $ChSim$, $SibSim$, $LeafSim$, and $PSim$ are similarity functions to compute the child, sibling, leaf, and ancestor context similarity between nodes respectively, and $combine_s$ is an aggregation function to combine these similarities.

$$SSimM[i,j] = combine_s(ChSim(n_i,n_j), SibSim(n_i,n_j), LeafSim(n_i,n_j), PSim(n_i,n_j)) \tag{9.3}$$

3. After computing both linguistic and structural similarities between Web service tree operations, we combine them. The output of this phase are $k \times k'$ total similarity matrices, $TSimM$. Equation 9.4 gives the entries of a matrix, where $combine$ is an aggregation function combining these similarities.

$$TSimM[i,j] = combine(LSimM[i,j], SSimM[i,j]) \tag{9.4}$$

Web Service Operation Similarity Matrix. We use $k \times k'$ total similarity matrices to construct the Web service operation similarity matrix, $OpSimM$. We compute the total similarity between every operation pairs by ranking element similarities in their total similarity matrix per element, selecting the best one, and averaging these selected similarities. Each computed value represents an entry in the matrix, $OpSimM[i,j]$, which represents the similarity between operation op_{1i} from the first set and operation op_{2j} from the second set.

Example 9.3 *Applying the proposed sequence-based matching approach to abstract parts illustrated in Fig. 9.4(b), we get* $OpSim(getData, getProduct) = 0.75$.

Algorithm complexity The worst case time complexity of the schema matching algorithm can be expressed as a function of the number of nodes in each operation, the number of operation in each WS, and the number of WSs. Let n be the average operation size, k be the average operation number, and S be the number of input WSs. Following the same process in Chapter 6, we can prove the overall time complexity of the algorithm as $O(n^2k^2)$.

9.5.3 Matching refinement

For every Web service pairs we have two sets of matrices: three $NSimM$ matrices that store the similarity between concrete elements, and one $OpSimM$ that stores the similarity between two web service operations. This provides the SeqDisc approach more flexibility in assessing the similarity between web services. As a consequence, we have two different possibilities to get the similarity:

1. *Using only abstract parts*; Given, the operation similarity matrix, $OpSimM$, that stores the similarity between operations of two web services, how to obtain the similarity between them. We can simply get the similarity between the two web services by averaging the similarity values in the matrix. However, this method produce smaller values, which do not represent the actual similarity among services. And due to uncertainty inherent in the matching process, the best matching can actually be an unsuccessful choice [66]. To overcome these shortcomings, similarity values are ranked up to top-2 ranking for each operation. Then, the average value is computed for these candidates. For example, let we have two web services. The first contains five operations ($k = 5$) and the number of operations in the second is four ($k' = 4$). And let after applying the proposed matching approach we get the operation similarity matrix shown below.
$$OpSim = \begin{pmatrix} 0.2 & 0.1 & 0.03 & 0.6 \\ 0.3 & 0.025 & 0.58 & 0.045 \\ 0.015 & 0.36 & 0.48 & 0.1 \\ 0.13 & 0.1 & 0.8 & 0.7 \\ 0.8 & 0.75 & 0.62 & 0.2 \end{pmatrix}$$
The similarity between the two web service has a value of 0.346 using the simple average function, while it has a value of 0.557 using the ranked method.

2. *Using both abstract and concrete parts*; The second possibility to assess the similarity between web services is to exploit both abstract and concrete parts. For any operation pair, $op_{1i} \in WSDL1$ and $op_{2j} \in WSDL2$, whose similarity is greater than a predefined threshold (i.e. $OpSimM[i, j] > th$), we increase the similarity of their corresponding parents (portType, binding, and port, respectively).

9.6 Experimental evaluation

To validate the performance of the SeqDisc approach, we present results obtained by conducting a set of experiments. In the following, we describe the data sets used to validate the approach throughout the evaluation, the performance criteria, and the experimental results.

9.6.1 Data sets

In order to evaluate the degree to which the SeqDisc approach can distinguish between Web services, we need to obtain families of related specifications. We found such a collection published by XMethods[8] and QWS data set [4]. We selected 78 WSDL documents from six different categories. Table 9.2 shows these categories and the number of Web services inside each one. Using the "analyze WSDL" method provided by XMethods, we identify the number of operations in each WS, and get the total number of operations inside each category, as shown in the table. All the experiments below share the same design: each service of the collection was used as the basis for the desired service; this desired service was then matched against the complete set to identify the best target service(s).

9.6.2 Performance Measure

We consider two performance aspects: the effectiveness and the efficiency of the approach. We use *precision, recall*, and *F-measure* to evaluate effectiveness, while the response time is used as a measure for efficiency. Precision (P) is the ability to provide the relevant Web services from a set of retrieved Web services, while recall (R) is the ability to provide the maximum number of Web services from a set of relevant Web services. F-measure is a trade-off between precision and recall. Mathematically, they are defined as follows:

$$P = \frac{|B|}{|A|}, R = \frac{|B|}{|C|}, F - measure = 2 \times \frac{P \times R}{P + R} \tag{9.5}$$

where $|B|$ is the number of returned relevant Web services, $|A|$ is the number of returned services, and $|C|$ is the number of relevant services. The response time is the time required to complete the task at hand, including both pre-processing and similarity measuring phases.

9.6.3 Experimental Results

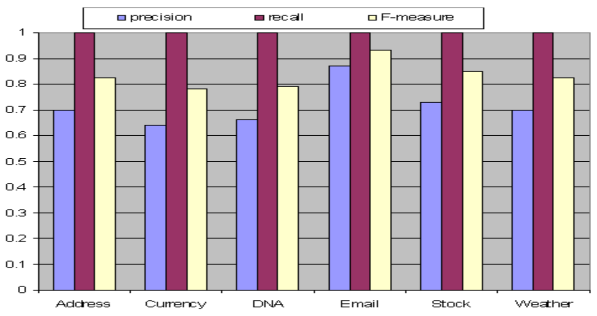
Effectiveness Evaluation

We have two possibilities to assess the similarity between Web services depending on the exploited information of WSDL specifications.

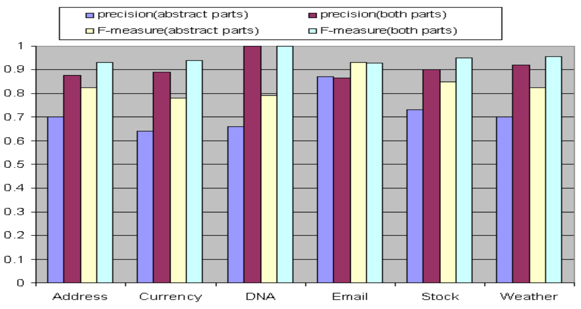
[8]http://www.xmethods.net

Table 9.2: Data set specifications

Category	No. of WSs	NO. of operations	Size (KB)
Address	13	50	360
Currency	11	88	190
DNA	16	48	150
Email	10	50	205
Stock quite	14	130	375
Weather	13	110	266



(a) Quality measures (abstract parts only)



(b) Quality measures (both parts)

Figure 9.5: Effectiveness evaluation of SeqDisc.

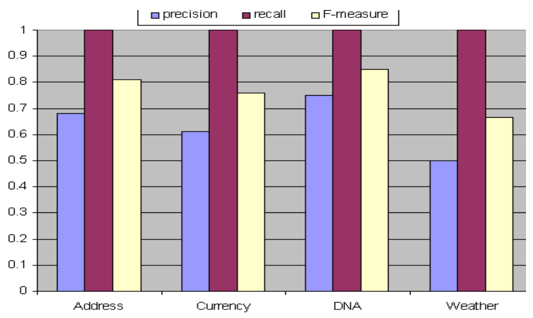
1. Assessing the Web service similarity using only abstract parts (operations).
   In the first set of experiments, we match abstract parts of each service tree from each category against the abstract parts of all other service trees from all categories. Then, we select a set of candidate services, such that the similarity between individual candidate services and the desired one is greater than a predefined threshold. Precision and recall are then calculated for each service within a category. These calculated values are averaged to determine the average precision and recall for each category. Precision, recall and F-measure are calculated for all categories and illustrated in Fig.9.5(a). There are several interesting findings, which are evident in this figure.

   First, the SeqDisc approach has the ability to discover all web services from a set of relevant services. As can be seen, across different six categories, the approach has a recall rate of 100% without missing any candidate service. This ability reflects the strong behavior of the approach of exploiting both semantic and structural information of WSDL specifications in an effective way. Second, the figure also shows that the ability of the approach to provide relevant web services from a set of retrieved services is reasonable. The precision of the approach across six categories ranges between 64% and 86%. This means that while the approach does not miss any candidate service, however, it produces false match candidates. This is due to the web service assessment approach is based on lightweight semantic information and does not use any external dictionary or ontology. Finally, based on precision and recall, our framework is almost accurate with F-measure ranging from 78% to 93%.

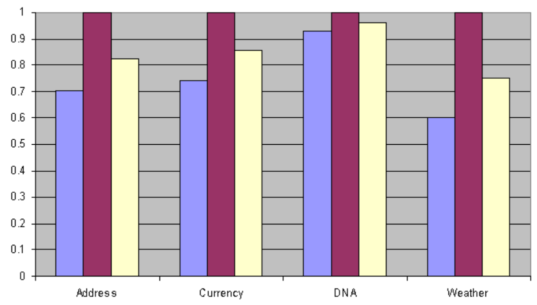
2. Assessing the Web service similarity using both abstract and concrete parts.
   The second set of experiments to assess the similarity between web services exploits the similarity between concrete parts as well as the similarity between their operations. In this set of experiments, we matched the whole parts (both abstract and concrete) of each service tree against all other service trees from all categories. Then, we selected a set of candidate services, such that the similarity between individual candidate services and the desired one is greater than a predefined threshold. Precision and recall are then calculated for each service within a category. These calculated values are averaged to determine the average precision and recall for each category. Precision and F-measure are calculated for all categories and illustrated in Fig. 9.5(b). We also compared them against the results of the first possibility. The results are reported in Fig. 9.5(b). The figure represents a number of appealing findings. (1) The recall of the approach remains at the unit level, i.e. no missing candidate services. (2) Exploiting more information about WSDL documents improves the approach precision, i.e. the number of false retrieved candidate services decreases across six different categories. The figure shows that the precision of the approach exploiting both concrete and abstract parts of service trees ranges between 86% in the Email category and 100% in the DNA category. (3) The first two findings lead to the quality of the approach is almost accurate with F-measure ranging between 90% and 100%.

Effect of Individual Matchers. We also performed another set of experiments to study the effect of individual matchers (linguistic and structure) on the effectiveness of web service similarity. To this end, we used data sets from the Address, Currency, DNA, and Weather domains. We consider the linguistic matcher utilizing either abstract parts or concrete and abstract parts. Figure 9.6 shows matching quality for these scenarios.

The results illustrated in Figure 9.6 show several interesting findings. (1) Recall of the SeqDisc approach has a value of 1 across the four domains either exploiting only abstract parts or exploiting both parts, as shown in Figure 9.6(a,b). This means that the approach



(a) Utilizing abstract parts.



(b) Utilizing both concrete and abstract parts.

Figure 9.6: Effectiveness evaluation of SeqDisc using linguistic matcher.

is able to discover the desired service even if the linguistic matcher is only used. (2) However, precision of the approach decreases across the tested domains (except only for the DNA domain using the abstract parts). For example, in the Address domain, precision decreases from 88% to 70% utilizing both parts, and it reduces from 92% to 60% utilizing both parts in the Weather domain. This results in low F-measure values compared with the results shown in Figure 9.5. (3) Exploiting both abstract and concrete parts outperforms exploiting only the abstract parts. This can be investigated by comparing results shown in Figure 9.6(a) to results in Figure 9.6(b).

To sum, using only the linguistic matcher is not sufficient to assess the similarity between web services. Hence, it is desirable to consider other matchers. As the results in Figure 9.5 indicates that the SeqDisc approach employing the structure matcher is sufficient to assess the similarity achieving F-measure between 90% and 100%.

Performance Comparison. Besides studying the performance of the SeqDisc approach, we also compared it with the discovery approach proposed in [36], called *KerDisc*[9]. To assess the similarity between the desired Web service (user query) and the available Web services, the KerDisc approach first extracts the content from the WSDL documents followed by stop-word removal & stemming [36]. The constructed support-based semantic kernel in the training phase is then used to find the similarity between WSDL documents and a query when the query is provided. The topics of WSDL documents which are most related to the query topics are considered to be the most relevant. Based on the similarity computed using the support-based semantic kernel, the WSDLs are ranked and a list of appropriate Web services is returned to the user.

Both SeqDisc and KerDisc have been validated using the data sets illustrated in Table

[9]We give the approach this name for easier reference.

9.2. The quality measures have been evaluated and results are reported in Figure 9.7. The figure shows that, in general, SeqDisc is more effective than KerDisc. It achieves higher F-measure than the other approach across five domains. It is worth noting that the KerDisc approach indicates low quality across the Address and Email domains. This results due to the two domains have common content, which produces many false positive candidates. The large number of false candidates declines the approach precision. Compared to the results of SeqDisc using only the linguistic matcher shown in Figure 9.6(b), our approach outperforms across the Address and DNA domains, while the KerDisc approach is better across the other domains. This reveals two interesting findings: (1) KerDisc can effectively locate the desired service among heterogeneous web service, while it fails to discover the desired service among a set of homogeneous services. In contrast, our approach could effectively locate the desired service among either a set of homogeneous or a set of heterogenous services. (2) SeqDisc clarifies the importance of exploiting the structure. matcher.

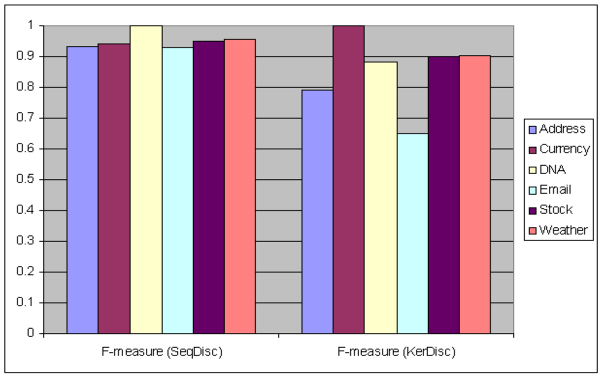
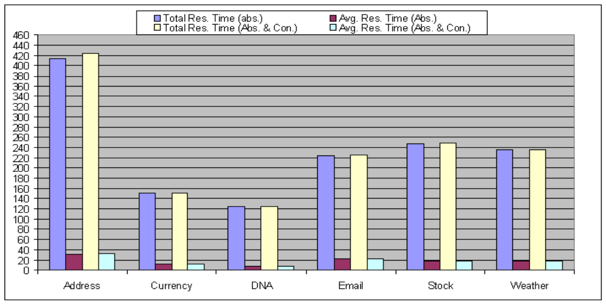


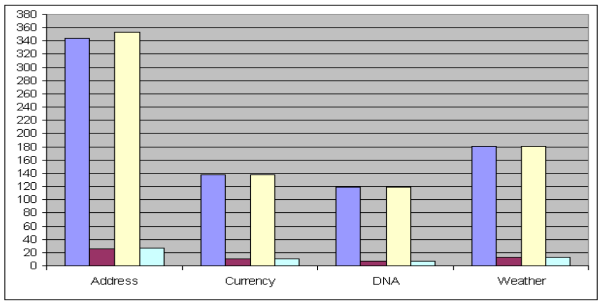
Figure 9.7: Effectiveness evaluation comparison.

Efficiency Evaluation

From the response time point of view, Figure 9.8(a) gives the response time that is required to complete the task at hand, including both pre-processing and similarity measuring phases. The reported time is computed as a total time and an average time. The total time is the time needed to locate desired Web services belonging to a certain category, while the average time is the time required to discover a Web service of the category. The figure also shows that the framework needs 124 seconds in order to identify all desired Web services in the DNA category, and it requires 7 seconds to discover one service in the category, while it needs 3.7 minutes to locate all services in the Email category. We



(a) SeqDisc evaluation.



(b) Using linguistic matcher.

Figure 9.8: Response time evaluation.

also considered the response time and compared it to the response time of the first set (i.e, using only the abstract parts). The results are calculated and listed in Figure 9.8(a). The figure shows that the response time required to locate the desired Web service using both abstract and concrete parts equals to the response time when only using abstract parts, or needs a few milliseconds more.

We also conducted another set of experiments to examine the effect of individual matchers on the response time. To this context, we used the same data sets used before in the section presenting the effect of individual matchers. Results are reported in Figure 9.8(b). The figure shows that using only the linguistic matcher consumes most time of the SeqDisc approach. For example, the approach needs 124 seconds to locate services inside the DNA domain, while it requires 118 seconds using only the linguistic matcher. This reveals that SeqDisc is benefit from representing web services using the Prüfer encoding method.

9.7 Summary

In this chapter we deployed our schema matching techniques in an important application domain: Web Service. As web services have emerged as one of distributed computing technologies, new round of interests from industrial and research communities have sparked. To be developed and reused, a web service relies on a set of related specifications, how to web services should be specified (through WSDL), how they should be advertised so that they can be discovered and accessed, and how they should be invoked at run time (through SAOP). This chapter addressed only the issue of effectively and efficiently locating web services.

To effectively and efficiently locate web service, in this chapter, we described a new and flexible approach to assess the similarity between Web services, which can be used to support a more automated Web service discovery framework. The approach makes use of the whole WSDL document specification and distinguishes between the concrete part and abstract parts. We started by analyzing web services represented by WSDL documents and modeled them as services trees. Then, we notice that concrete parts from different Web services have the same hierarchal structure, hence we devised a level matching approach. While the abstract parts have different structures, therefore, we developed a sequence-based schema matching approach to compute the similarity between them.

We have developed and implemented algorithms and methodologies that realize our approach. To validate the performance of the approach, we have conducted two sets of experiments depending on the exploited information of services trees. Two performance aspects have been measured: the quality and the efficiency of the approach. Our experimental results have shown that our method is accurate and scale-well. In particular, exploiting both (concrete and abstract) parts in assessing the similarity between web services outperforms exploiting only abstract parts. In the first case, the approach has F-measure ranging between 90% and 100%, while in the second between 78% and 93%. However and in fact, we are still a long way from automatic service discovery. This work has been done to assess our schema matching techniques.

Part V

Summary and future directions

10 Conclusions

This dissertation investigated the problem of XML schema matching. XML has recently emerged as a major means for information exchange on the Web and within organizations, and has been broadly used for information representation. Consequently, XML databases on the Web are proliferating. To enable and support the interoperability and exchange of data, there has been a growing need to identify and discover semantically similar elements across different data sources. Discovering simple mappings partially solves the schema matching problem. However, discovering complex mappings increases the difficulties of the problem, especially when dealing with large-scale schemas. To this context, we proposed a new XML schema matching approach, called sequence-based schema matching, discussed its technical details and its evaluation. The main findings of each chapter of the thesis are summarized one by one in sequel, while the contributions of the thesis are reported in the other section.

10.1 Summary of the thesis

We classified the dissertation in five cooperative parts. **Part I** is devoted to state the schema matching problem, giving the different dimensions of the problem and objectives of the dissertation, **Part II** is dedicated to present the state-of-the-art in the schema matching problem, **Part III** is committed to introduce the sequence-based schema matching approach and its evaluation, **Part IV** is dedicated to show how to deploy the proposed approach in several application domains, and the final **Part V** is deployed to give a summary of the thesis contributions and its future work.

Part I is divided into two chapters. Chapter 1 outlined our motivations to investigate the XML schema matching problem. Furthermore, it presented an abstract road map of the dissertation. Chapter 2 introduced an (in)formal definition of the schema matching problem, including problem statement, input data models, and the matching output. It focused on different data models that represent input schemas, and paid more attention to XML schemas identifying what different heterogeneities are, what issues in representing large-scale schemas are and why they increase the complexity of schema matching.

Part II also included two chapters. In Chapter 3, we motivated the importance of schema matching by reporting its use in several application domains. We showed that a common feature of these applications is that their XML schemas are independently developed resulting in semantic and structure heterogeneities. Furthermore, we pointed out how these applications make use of schema matching techniques during either design time or run time. In Chapter 4, we introduced a unified schema matching framework in order to discuss and report on existing schema matching systems. We could differentiate between these systems based on their input schemas, methodologies to identify corresponding elements, and their matching results. The aim was to draw a clear picture on the approaches, how far we can go with the current solutions, and what remains to be achieved. Here, the focus is on the concepts, alternatives, and fundamentals of the solutions, not on detailed technical discussions.

Part III encompassed three chapters. In Chapter 5, we introduced the definition, adoption, and utilization of XML element similarity measures that can be used to assess the similarity between XML schema elements. We gathered both the element features and the element relationships while measuring their similarity. However, it has been shown that representing XML schemas using schema trees (graphs) is not sufficient to achieve our objectives. Therefore, we made use of the Prüfer encoding method that constructs a one-to-one correspondence between schema trees and Prüfer sequences. This representation provides the ability to exploit the semantic information of schema trees in Label Prüfer sequences (LPS), and to exploit the structural information in Number Prüfer Sequences (NPS). Furthermore, it helps to exploit of these information as quickly as possible. We explained the relationships between the sequence representation and the element features and element contexts.

In Chapter 6 we presented the sequence-based schema matching approach, introducing several element similarity measure algorithms. We developed a new matching system, called *XPrüM*, to identify and discover simple mappings in the context of large-scale XML schema. To deal with complex mappings, we extended the original system and developed a new matching system, called *EXPrüM*. We innovated the concept of compatible elements to reduce the search space. The complexity analysis of both systems has been studied and it has been shown that *EXPrüM* has a lower time complexity compared to the original system. In Chapter 7 we started with discussing different criteria that can be used to evaluate the performance of schema matching introducing a new criterion, called *cost-effectiveness*, that combines both matching effectiveness and matching efficiency. We then carried out three sets of experiments to evaluate and validate our proposed approaches. The first set is to evaluate XML element similarity measures using different settings and different combining strategies. The other two sets of experiments have been guided by the results obtained from the first set. The second set is used to validate the *XPrüM* system, while the third set is to evaluate the enhanced system.

Part V involved two chapters. In Chapter 8, we showed how to deploy the sequence-based matching approach to cluster a set of heterogeneous XML schemas. To this end, we developed a new clustering framework, called *SeqXClust*. In Chapter 9, we applied the matching approach to another application domain. We used the matching approach to assess the similarity between a set of Web services in order to automatically support the discovery of desired Web service. To this context, we developed a new Web service assessment framework, called *SeqDisc*. Both frameworks have been developed, implemented, and validated using real-world data sets.

10.2 Contributions

This dissertation introduces several contributions. We categorize the set of our contributions as follows:

1. State-of-the-art;
    - introducing a detailed and up-to-date survey of schema matching approaches and systems under a unified framework.
    - pointing out different application domains that can make use of schema matching.
2. Methodology;
    - introducing a set of XML element similarity measures based on either element features or element relationships.

- building a conceptual connection between the schema matching problem and the Prüfer encoding method.
- proposing a new representation of XML schema trees using the Prüfer encoding method.
- introducing a new strategy to combine element similarity measures.

3. Development and implementation;

    - designing, developing, and implementing the *XPrüM* system to cope with simple mappings in the context of large-scale schemas.
    - designing, developing, and implementing the *EXPrüM* to cope with both simple and complex mappings in the context of large-scale schemas.

4. Deployment;

    - designing, developing, and implementing the *SeqXClust* to show the deployment of proposed matching approaches in XML schema clustering.
    - designing, developing, and implementing the *SeqDisc* to point out how to utilize our matching approaches in Web service discovery.

5. Evaluation;

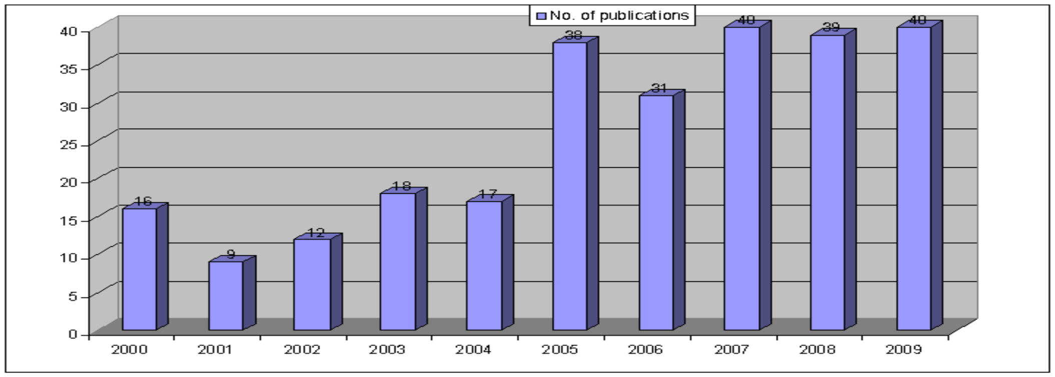
    - introducing the *cost-effectiveness* measure to combine both matching performance aspects.
    - conducting intensive sets of experiments to validate and evaluate our proposed systems and frameworks using several real-world data sets.
    - carrying out a set of comparisons with well-known and available schema matching systems to validate the performance of our systems and frameworks.

11 Future directions

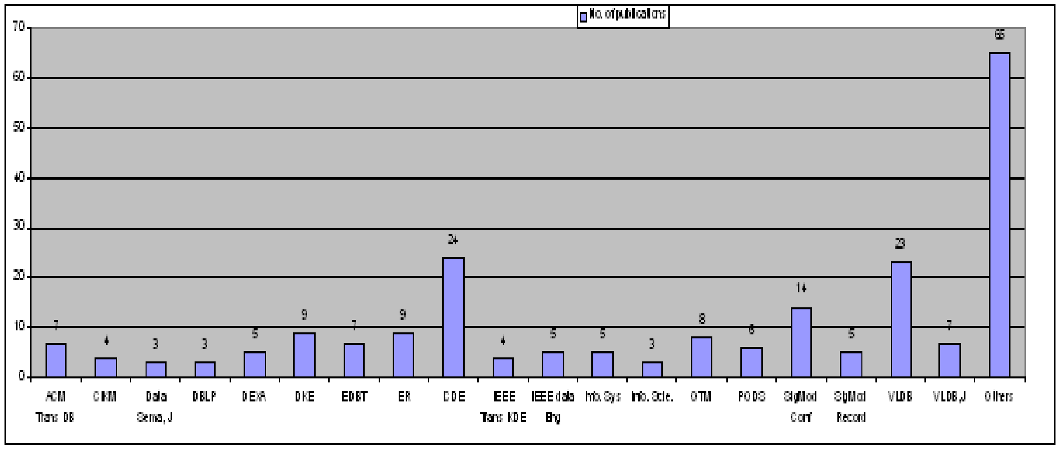
The schema/ontology matching problem is still an open research problem. In the near future, we expect an ongoing increase of works on schema/ontology matching. This expectation is build on the study of the distributions of many works devoted to diverse aspects of schema matching and published at various conferences and journals. Using the arnetminer search engine[1], we study the effort devoted to schema matching w.r.t. either the number of publications per year or the number of publications appeared in specific conferences and journals. A result of this study is reported in Fig. 11.1. The figure points out two interesting findings: (i) There is an increasing effort dedicated to schema matching during this decade, as shown in Fig. 11.1a. (ii) Due to the arising importance of the schema matching topic a large number of matching publications have appeared in well-known and respective conferences and journals, as shown in Fig. 11.1b.

In this chapter, we discuss the ongoing and future works on schema matching along the following three directions: (1) methodologies used to assess the similarity between schema elements, (2) prototypes developed to implement matching methodologies, and (3) evaluation used to validate the matching performance.

[1] http://www.arnetminer.org/



(a) Publication distribution per year.



(b) Publication distribution over conferences and journals.

Figure 11.1: Publications focusing on schema matching.

11.1 Methodology directions

This direction concerns the methodologies that are used to assess and combine the similarity between schema elements. In general, these methodologies can be divided into three categories: *pairwise*, *incremental*, or *holistic*. Pairwise schema matching identifies and discovers semantic correspondences between two schemas at a time; the methodology used to ascertain these correspondences between a mediated schema and a set of local schemas is called incremental schema matching, while holistic schema matching determines similar elements among a set of schemas. Holistic schema matching is only used to discover correspondences among web query interfaces, incremental schema matching is used for mediated schema construction, and pairwise schema matching is the generic matching methodology. Therefore, further investigations have to be devoted to the other methods. In particular, the following issues should be considered.

- **Element similarity measure (matcher)**; It has been shown that to effectively assess the similarity between schema elements, both element features and element relationships should be exploited. As a result, there exist element measures (matchers) to quantify its similarity using element features and others using element relationships. However, most of the existing matchers are based on Information Retrieval approaches. There is a growing need to propose and develop schema matching-specific matchers.

- **Uncertainty in element matcher**; The semantics of the involved elements can be inferred from only a few information sources, typically the data creators, documentation, and associated schema and data [51]. Extracting semantics information from these sources is often extremely bulky. Furthermore, element measures are based on clues in the schema and data, such as element names, types, data values, schema structures, and integrity constraints. However, these clues are often unreliable and incomplete. Therefore, the result of schema matching used in real-world applications is often uncertain. To handle these uncertainties, several works have been proposed [66, 57, 97, 67, 40]. However, more work is needed to fully automated uncertainty in schema matching.

- **Element similarity measure combining**; It has been shown that using a single element similarity measure is not sufficient to assess the similarity between schema elements. This necessitates the need to utilize several element measures exploiting both internal element features and external element relationships. Utilizing several element measures provides the advantage of matching algorithms to be more flexible. However, it also embeds a disadvantage of how to combine these similarity measures. Most of the existing matching systems make use of the linear combining strategy. In this thesis, we utilized the (fixed) nonlinear combining strategy. However, this strategy needs more investigations, especially the adaptive nonlinear method.

- **Element similarity selector**; It has been shown that combining element measures cannot guarantee that the optimal mapping will always be identified by picking up the correspondence with the highest score. To overcome such situations, the approach in which K (and not just one) top-ranked schema mappings can be adopted [55]. New schema matching-specific algorithms are needed to address the correct mapping selection.

11.2 Prototype directions

This direction is devoted to the implementation details of the methodologies mentioned above. In particular, we address the following issues.

- **GUI**; Here, we should distinguish between research and commercial prototypes and systems. Commercial systems, such as BizTalk schema mapper[2] and Altova MapForce[3] are interested in developing attractive Graphical User Interfaces (GUI) to make the matching process more easier for the end user. However, the main concern of research systems is to automate the matching process. However, some of research systems, such as COMA++[4] provide their matching prototypes with GUIs. More efforts are needed in this direction to address the necessary requirements of schema matching tools to be more generic and attractive.

- **User context**; It has been shown that fully automated schema matching is infeasible, specifically when discovering complex matches [138]. The arising task is to accurately identify the critical points where user input is maximally useful. Another related issue is how to map the user context information to input schema elements.

[2]http://www.microsoft.com/biztalk/en/us/roadmap.aspx
[3]http://www.altova.com/mapforce.html
[4]http://dbserv2.informatik.uni-leipzig.de/coma/

11.3 Performance & evaluation directions

- **Performance**; Schema matching is a complex process. The shift from manual schema matching done by human experts, to automatic matching using various element similarity measures (schema matchers), and from the small-scale scenario to the large-scale scenario increases the complexity of schema matching. In this context, we should deal with two schema matching performance aspects. Matching quality (effectiveness) is a crucial aspect in automatic schema matching, while matching efficiency (scalability) is important in large-scale scenarios. Therefore, developing new schema matching approaches should balance between the two matching performance aspects.

- **Evaluation**; Not only is the performance of schema matching still an open research challenge, but also the methodology used to evaluate this performance. Almost all existing matching systems deal with small-scale schemas, and hence, they make use of effectiveness-based criteria to evaluate their systems [46]. Introducing large-scale schemas to schema matching necessitates the need for additional criteria to evaluate the matching performance. The response time has recently been used as a measure for matching efficiency, and a new benchmark for schema matching has been developed [59]. However, both aspects have been evaluated separately. An attempt to combine the performance criteria has been proposed in [9]. However, new methods that quantify both pre- and post-match effort are needed.

Bibliography

[1] Data integration glossary. In *U.S. Department of Transportation*, pages 1–18. USA, 2001.

[2] S. Abiteboul, D. Suciu, and P. Buneman. *Data on the Web: From Relations to Semistructed Data and XML*. Morgan Kaumann, USA, 2000.

[3] C. C. Aggarwal, N. Ta, J. Wang, J. Feng, and M. J. Zaki. Xproj: a framework for projected structural clustering of XML documents. In *KDD 2007*, pages 46–55, 2007.

[4] E. Al-Masri and Q. Mahmoud. Qos-based discovery and ranking of web services. In *ICCCN 2007*, pages 529 – 534, 2007.

[5] A. Algergawy, R. Nayak, and G. Saake. XML schema element similarity measures: A schema matching context. In *8th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE 2009) at OTM Conferences Part II Lecture Notes in Computer Science 5871*, pages 1246–1253. Vilamoura, Portugal, Nov. 2009.

[6] A. Algergawy, R. Nayak, E. Schallehn, and G. Saake. Supporting web service discovery by assessing web service similarity. In *13th East-European Conference on Advances in Databases and Information Systems (ADBIS-2009)*. Riga, Latvia, Sept. 2009.

[7] A. Algergawy and G. Saake. A classification scheme for XML data clutering techniques. In *4th International Conference on Intelligent Computing and Information Systems (ICICIS 2009)*, pages 550–555. Cairo, Egypt, March 2009.

[8] A. Algergawy, E. Schallehn, and G. Saake. A unified schema matching framework. In *19. GI-Workshop on Foundations of Databases*, pages 58–62. Bretten, Germany, May 2007.

[9] A. Algergawy, E. Schallehn, and G. Saake. Combining effectiveness and efficiency for schema matching evaluation. In *First International Workshop on Model-Based*

Software and Data Integration (MBSDI 2008), volume 8 of *CCIS, Springer*, pages 19–30. Berlin, Germany, April 2008.

[10] A. Algergawy, E. Schallehn, and G. Saake. Fuzzy constraint-based schema matching formulation. In *Business Information Systems (BIS 2008) Workshops*, pages 141–152, Innsbruck, Austria, 2008. CEUR Workshop Proceedings 333.

[11] A. Algergawy, E. Schallehn, and G. Saake. Fuzzy constraint-based schema matching formulation. *Scalable Computing: Practice and Experience, Special Issue: The Web on the Move*, 9(4):303–314, Dec. 2008.

[12] A. Algergawy, E. Schallehn, and G. Saake. A Prufer sequence-based approach for schema matching. In *Eighth International Baltic Conference on Databases and Information Systems (BalticDB&IS2008)*, pages 205–216. Estonia, June 2008.

[13] A. Algergawy, E. Schallehn, and G. Saake. A schema matching-based approach to XML schema clustering. In *The Tenth International Conference on Information Integration and Web-based Applications Services (iiWAS)*, pages 131–136. Linz, Austria, ACM, 2008.

[14] A. Algergawy, E. Schallehn, and G. Saake. A sequence-based ontology matching approach. In *Proceedings of the Fourth International Workshop on Contexts and Ontologies (C&O) Collocated with the 18th European Conference on Artificial Intelligence (ECAI-2008)*, pages 26–30. Patras, Greece, July 2008.

[15] A. Algergawy, E. Schallehn, and G. Saake. *Databases and Information Systems V - Selected Papers from the Eighth International Baltic Conference*, volume 187, chapter A New XML Schema Matching Approach Using Prüfer Sequences, pages 217–228. ISO Press, 2009.

[16] A. Algergawy, E. Schallehn, and G. Saake. Efficiently locating web services using a sequence-based schema matching approach. In *11th International Conference on Enterprise Information Systems (ICEIS)*, pages 287–290. Milan, Italy, May 2009.

[17] A. Algergawy, E. Schallehn, and G. Saake. Improving XML schema matching performance using Prüfer sequences. *Data & Knowledge Engineering*, 68(8):728–747, August 2009.

[18] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In *EDBT'02*, pages 89–102, 2002.

[19] C. Atkinson, P. Bostan, O. Hummel, and D. Stoll. A practical approach to web service discovery and retrieval. In *ICWS 2007*, pages 241–248, 2007.

[20] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, pages 223–240. Kyoto, Japan, 1989.

[21] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *ACM SIGMOD International Conference on Management of Data*, pages 906–908, 2005.

[22] A. Avila-rosas, L. Moreau, V. Dialani, S. Miles, and X. Liu. Agents for the grid: A comparison with web services (part ii: Service discovery). In *AAMAS02*, pages 52–56, 2002.

[23] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press/Addison-Wesley, 1999.

[24] M. Banek, B. Vrdoljak, A. M. Tjoa, and Z. Skocir. Automating the schema matching process for heterogeneous data warehouses. In *9th International Conference on Data Warehousing and Knowledge Discovery*, pages 45–54, 2007.

[25] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput Surv*, 18(4):323–364, 1986.

[26] G. Beliakov, A. Pradera, and T. Calvo. *Aggregation Functions: A Guide for Practitioners*, volume 221 of *Studies in Fuzziness and Soft Computing*. Springer, 2007.

[27] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. *SPIRE*, pages 39–48, 2004.

[28] P. Berkhin. *Grouping Multidimensional Data: Recent Advances in Clustering*, chapter Survey of Clustering Data Mining Techniques, pages 25–71. Springer Berlin Heidelberg, 2006.

[29] P. A. Bernstein, T. Bergstraesser, J. Carlson, S. Pal, P. Sanders, , and D. Shutt. Microsoft repository version 2 and the open information model. *Information Systems*, 24(2):71–98, 1999.

[30] P. A. Bernstein, A. Y. Halevy, and R. A. Pottinger. A vision for management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.

[31] P. A. Bernstein and E. Rahm. Data warehouse scenarios for model management. In *19th International Conference on Conceptual Modeling*, pages 1–15, 2000.

[32] E. Bertino and E. Ferrari. XML and data integration. *IEEE Internet Computing*, 5(6):75–76, 2001.

[33] D. Bianchini, V. Antonellis, and M. Melchiori. Flexible semantic-based service matchmaking and discovery. *World Wide Web*, 11(2):227–251, 2008.

[34] A. Bonifati, G. Mecca, A. Pappalardo, and S. Raunich. The spicy project: A new approach to data matching. In *SEBD*. Turkey, 2006.

[35] A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa. Schema mapping verification: the spicy way. In *11th International Conference on Extending Database Technology (EDBT2008)*, pages 85–96. Nantes, France, 2008.

[36] A. Bose, R. Nayak, and P. Bruza. Improving web service discovery by using semantic models. In *WISE 2008*, pages 366–380. New Zealand, 2008.

[37] A. Boukottaya and C. Vanoirbeek. Schema matching for transforming structured documents. In *DocEng'05*, pages 101–110, 2005.

[38] L. Cabral, J. Domingue, E. Motta, T. R. Payne, and F. Hakimpour. Approaches to semantic web services: an overview and comparisons. In *ESWS 2004*, pages 225–239. Greece, 2004.

[39] D. Carmel, N. Efraty, G. M. Landau, Y. S. Maarek, and Y. Mass. An extension of the vector space model for querying XML documents via XML fragments. *SIGIR Forum*, 36(2), 2002.

[40] R. Cheng, J. Gong, and D. W. Cheung. Managing uncertainty of XML schema matching. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2009.* California, USA, 2010.

[41] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[42] P. Cudr-Mauroux, S. Agarwal, and K. Aberer. Gridvine: An infrastructure for peer information management. *IEEE Internet Computing*, 11(5):36–44, 2007.

[43] P. Cudr-Mauroux, S. Agarwal, A. Budura, P. Haghani, and K. Aberer:. Self-organizing schema mappings in the GridVine peer data management system. In *the 33rd International Conference on Very Large Data Bases, VLDB2007*, pages 1334–1337, 2007.

[44] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering complex semantic matches between database schemas. In *SIGMOD Conference 2004*, pages 383–394, 2004.

[45] P. F. Dietz. Maintaining order in a linked list. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 122 – 127, 1982.

[46] H. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *the 2nd Int. Workshop on Web Databases*, 2002.

[47] H. H. Do and E. Rahm. COMA- a system for flexible combination of schema matching approaches. In *VLDB 2002*, pages 610–621, 2002.

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

[49] A. Doan. Learning to map between structured representations of datag. In *Ph.D Thesis*. Washington University, 2002.

[50] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. *SIGMOD*, pages 509–520, May 2001.

[51] A. Doan and A. Halevy. Semantic integration research in the database community: A brief survey. *AAAI AI Magazine, Special Issues on semantic Integration*, 25(1):83–94, 2005.

[52] A. Doan, Y. Lu, Y. Lee, and J. Han. Profile-based object matching for information integration. *IEEE Intelligent Systems*, 18(5):54–59, 2003.

[53] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. *Ontology matching: A machine learning approach.* Handbook on Ontologies, International Handbooks on Information Systems, 2004.

[54] A. Doan, J. Madhavan, P. Domingos, and A. Y. Halevy:. Learning to map between ontologies on the semantic web. In *the Eleventh International World Wide Web Conference*, pages 662–673. USA, 2002.

[55] C. Domshlak, A. Gal, and H. Roitman. Rank aggregation for automatic schema matching. *IEEE on KDE*, 19(4):538–553, April 2007.

[56] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *VLDB 2004*, pages 372–383. Canada, 2004.

[57] X. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. In *VLDB'07*, pages 687–698, 2007.

[58] C. Drumm, M. Schmitt, H.-H. Do, and E. Rahm. Quickmig - automatic schema matching for data migration projects. In *the Sixteenth ACM Conference on Information and Knowledge Management, CIKM07*, pages 107–116. Portugal, 2007.

[59] F. Duchateau, Z. Bellahsene, and E. Hunt. Xbenchmatch: a benchmark for XML schema matching tools. In *VLDB 2007*, pages 1318–1321. Austria, 2007.

[60] F. Duchateau, Z. Bellahsene, and M. Roche. A context-based measure for discovering approximate semantic matching between schema elements. In *RCIS 2007*, pages 9–20. Morocco, 2007.

[61] F. Duchateau, Z. Bellahsene, and M. Roche. An indexing structure for automatic schema matching. In *SMDB Workshop*. Turkey, 2007.

[62] M. Ehrig and S. Staab. QOM- quick ontology mapping. In *International Semantic Web Conference*, pages 683–697, 2004.

[63] J. Euzenat and et al. State of the art on ontology alignment. In *Part of research project funded by the IST Program, Project number IST-2004-507482*. Knowledge Web Consortim, 2004.

[64] D. Florescu and D. Kossmann. Storing and querying XML data using an RDMBS. *IEEE Data Eng. Bull.*, 22(3):27–34, 1999.

[65] A. Gal. On the cardinality of schema matching. In *OTM Workshops*, pages 947–956, 2005.

[66] A. Gal. Managing uncertainty in schema matching with top-k schema mappings. *Journal on Data Semantics*, 6:90–114, 2006.

[67] A. Gal, M. V. Martinez, G. I. Simari, and V. S. Subrahmanian. Aggregate query answering under uncertain schema mappings. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009*, pages 940–951. Shanghai, China, 2009.

[68] A. Gal, A. Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, 14(1):50–67, 2005.

[69] J. Garofalakis, Y. Panagis, E. Sakkopoulos, and A. Tsakalidis. Web service discovery mechanisms: Looking for a needle in a haystack? In *International Workshop on Web Engineering*. Santa Cruz, 2004.

[70] F. Giunchiglia, M. Yatskevich, and P. Shvaiko. Semantic matching: algorithms and implementation. *Journal on Data Semantics*, 9:1–38, 2007.

[71] G. Gou and R. Chirkova. Efficiently querying large XML data repositories: A survey. *IEEE Trans. on Knowledge and Data Engineering*, 19(10):1381–1403, 2007.

[72] G. Guerrini, M. Mesiti, and D. Rossi. Impact of XML schema evolution on valid documents. In *Seventh ACM International Workshop on Web Information and Data Management (WIDM 2005)*, pages 39–44, 2005.

[73] G. Guerrini, M. Mesiti, and I. Sanz. *An Overview of Similarity Measures for Clustering XML Documents. Web Data Management Practices: Emerging Techniques and Technologies*. IDEA GROUP, 2007.

[74] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[75] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The piazza peer data management system. *IEEE Trans. Knowl. Data Eng.*, 16(7):787–798, 2004.

[76] Y. Hao and Y. Zhang. Web services discovery based on schema matching. In *ACSC2007*, pages 107–113. Australia, 2007.

[77] B. He and K. C.-C. Chang. Automatic complex schema matching across web query interfaces: A correlation mining approach. *ACM Trans. on Database Systems*, 31(1):346–395, 2006.

[78] J. Hopfield and D. Tank. Neural computation of decisions in optimization problems. *Biol Cybern.*, 52(3):52–141, 1985.

[79] Z. Hu. Using ontology to bind web services to the data model of automation systems. In *Web, Web-Services, and Database Systems*, pages 154–168, 2002.

[80] Y. B. Idrissi and J. Vachon. Evaluation of hierarchical clustering algorithms for document datasets. In *the 11th International Conference on Information and knowledge Management*, pages 515–524, 2002.

[81] Y. B. Idrissi and J. Vachon. A context-based approach for the discovery of complex matches between database sources. In *DEXA 2007, LNCS 4653*, page 864873, 2007.

[82] A. Jain, M. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[83] G. Kappel, E. Kapsammer, S. Rausch-Schott, and W. Retschitzegger. X-ray: Towards integrating XML and relational database systems. In *Proceedings of the 19th International Conference on Conceptual Modeling (ER2000)*, pages 339–353. Kyoto, Japan, 2000.

[84] T. Kohonen. *Self-Organizing Maps*, volume 30. Springer Series in Information Sciences, Berlin, Germany, 3 edition, 2001.

[85] N. Kokash. A comparison of web service interface similarity measures. In *the Third Starting AI Researchers' Symposium (STAIRS 2006)*, pages 220–231. Trentino, Italy, 2006.

[86] D. Kramer. Xem: XML evolution management. Master's thesis, Worchester Polytechnic Institute, 2001.

[87] D. X. Le, J. W. Rahayu, and E. Pardede. Dynamic approach for integrating web data warehouses. In *ICCSA 2006*, pages 207–216, 2006.

[88] D. Lee and W. W. Chu. Comparative analysis of six XML schema languages. *SIGMOD Record*, 29(3):76–87, 2000.

[89] M. L. Lee, L. H. Yang, W. Hsu, and X. Yang. Xclust: Clustering XML schemas for effective integration. In *CIKM'02*, pages 63–74, 2002.

[90] Y. Lee, M. Sayyadian, A. Doan, and A. Rosenthal. etuner: tuning schema matching software using synthetic scenarios. *VLDB J.*, 16(1):97–132, 2007.

[91] M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, page 233246, 2002.

[92] H.-P. Leung, F.-L. Chung, and S. C.-F. Chan. On the use of hierarchical information in sequential mining-based XML document similarity computation. *Knowledge and Information Systems*, 7(4):476–498, 2005.

[93] W. Li and C. Clifton. Semint: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data and Knowledge Engineering*, 33:49–84, 2000.

[94] W. Lian, D. W. lok Cheung, N. Mamoulis, and S.-M. Yiu. An efficient and scalable algorithm for clustering XML documents by structure. *IEEE Trans. on KDE*, 16(1):82–96, 2004.

[95] J. Ma, Y. Zhang, and J. He. Efficiently finding web services using a clustering semantic approach. In *CSSSIA 2008*, page 5. China, 2008.

[96] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB 2001*, pages 49–58. Roma, Italy, 2001.

[97] A. Marie and A. Gal. Boosting schema matchers. In *Proceedings of the OTM 2008 Confederated International Conferences*, pages 283 – 300, 2008.

[98] R. Marler and J. arora. Survey of multi-objective optimization methods for engineering. *Struct Multidisc Optim*, 26:369–395, 2004.

[99] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[100] W. Meier. eXist: An open source native XML database. In *Web, Web Services, and Database Systems*, pages 169–183. Erfurt, Germany, 2002.

[101] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, 2002.

[102] S. Melnik, E. Rahm, and P. A. Bernstein. Rondo: A programming platform for generic model management. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 1193–204. California, USA, 2003.

[103] R. J. Miller, L. M. Haas, and M. A. Hernndez. Schema mapping as query discovery. In *Proc 26th Int Conf On Very Large Data Bases (VLDB)*, page 7788, 2000.

[104] H. J. Moon, C. Curino, H. MyungWon, and C. Zaniolo. PRIMA: Archiving and querying historical data with evolving schemas. In *SIGMOD Conference*, 1019-1021 2009.

[105] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[106] R. Nayak. Fast and effective clustering of XML data using structural information. *Knowledge and Information Systems*, 14(2):197–215, 2008.

[107] R. Nayak. Using data mining in web services planning, development and maintenance. *International Journal of Web Services Research*, 5:62–80, 2008.

[108] R. Nayak and W. Iryadi. XML schema clustering with semantic and hierarchical similarity measures. *Knowledge-based Systems*, 20:336–349, 2007.

[109] R. Nayak and B. Lee. Web service discovery with additional semantics and clustering. In *IEEE/WIC/ACM International Conference on Web Intelligence, WI2007*, pages 555 – 558, 2007.

[110] N. Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.

[111] T. Özsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 1999.

[112] J. Pei, J. Hong, and D. A. Bell. A novel clustering-based approach to schema matching. In *4th International Conference Advances in Information Systems (ADVIS)*, pages 60–69, 2006.

[113] C. Pluempitiwiriyawej and J. Hammer. Element matching across data-oriented XML sources using a multi-strategy clustering model. *Data & Knowledge Engineering*, 48:297–333, 2004.

[114] R. Pottinger and P. A. Bernstein. Creating a mediated schema based on initial correspondences. *IEEE Data Eng. Bull.*, 25(3):26–31, 2002.

[115] H. Prufer. Neuer Beweis eines Satzes uber Permutationen. *Archiv für Mathematik und Physik*, 27:142–144, 1918.

[116] J. W. Rahayu, E. Pardede, and D. Taniar. The new era of web data warehousing: XML warehousing issues and challenges. In *The Tenth International Conference on Information Integration and Web-based Applications Services*, page 4, Nov. 2008.

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

[118] E. Rahm, H. H. Do, and S. Massmann. Matching large XML schemas. *SIGMOD Record*, 33(4):26–31, 2004.

[119] A. Renner. XML data and object databases: the perfect couple? In *17th Inter. Conf. on Data Engineering*, pages 143–148, 2001.

[120] R. Rojas. *Neural Networks - A Systematic Introduction*. Springer Verlag, Berlin, New York, 1996.

[121] A. Sajjanhar, J. Hou, and Y. Zhang. Algorithm for web services matching. In *APWeb 2004, LNCS 3007*, pages 665–670, 2004.

[122] B. Saleem, Z. Bellahsene, and E. Hunt. PORSCHE: Performance oriented schema mediation. *Information Systems*, 33(7-8):637–657, 2008.

[123] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *25th VLDB*, pages 302–314, 1999.

[124] P. Shvaiko. *Iterative Schema-based Semantic Matching*. PhD thesis, University of Trento, 2006.

[125] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *J. Data Semantics*, 4:146–171, 2005.

[126] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[127] M. Smiljanic. *XML Schema Matching Balancing Efficiency and Effectiveness by means of Clustering*. PhD thesis, Twente University, 2006.

[128] M. Smiljanic, M. van Keulen, , and W. Jonker. Formalizing the XML schema matching problem as a constraint optimization problem. In *16th International Conference on Database and Expert Systems Applications (DEXA2005)*, pages 333–342, 2005.

[129] M. Smiljanic, M. van Keulen, and W. Jonker. Using element clustering to increase the efficiency of XML schema matching. In *ICDE Workshops 2006*, pages 45–54, 2006.

[130] S. Staab and R. Studer. *Handbook of ontologies. International handbooks on information systems*. Springer Verlag, Berlin, Germany, 2004.

[131] R. Steinmetz and K. (Eds.). *Peer-to-Peer Systems and Applications*. Lecture Notes in Computer Science No. 3485. Springer Verlag, Berlin, Heidelberg, 2005.

[132] E. Stroulia and Y. Wang. Structural and semantic matching for assessing web-service similarity. *International Journal of Cooperative Information Systems (IJCIS)*, 14(4):407–438, 2005.

[133] S. Tatikonda, S. Parthasarathy, and M. Goyder. LCS-TRIM: Dynamic programming meets XML indexing and querying. In *VLDB'07*, pages 63–74, 2007.

[134] J. Tekli, R. Chbeir, , and K. Yetongnon. An overview on XML similarity: background, current trends and future directions. *Computer Science Review*, 3(3):151–173, 2009.

[135] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2), 1996.

[136] P. Valduriez and E. Pacitti. Data management in large-scale P2P systems. In *6th International Conference of Vector and Parallel Processing (VECPAR)*, pages 104–118, 2004.

[137] B. Vrdoljak, M. Banek, and Z. Skocir. Integrating XML sources into a data warehouse. In *Data Engineering Issues in E-Commerce and Services, Second International Workshop, DEECS 2006*, pages 133–142, 2006.

[138] G. Wang, J. A. Goguen, Y.-K. Nam, and K. Lin:. Critical points for interactive schema matching. In *Proceedings of 6th Asia-Pacific Web Conference, APWeb*, pages 654–664. Hangzhou, China, 2004.

[139] G. Wang, B. Sun, J. Lv, and G. Yu. RPE query processing and optimization techniques for XML databases. *J. Comput. Sci. Technol.*, 19(2):224–237, 2004.

[140] Y. Wang and E. Stroulia. Flexible interface matching for web-service discovery. In *WISE 2003*, pages 147–156. Italy, 2003.

[141] I. W. S. Wicaksana and K. Ytongnon. A peer-to-peer based semantic agreement approach for information systems interoperability. In *OM-2006 ISWC-2006*, pages 216–220, 2006.

[142] B. Y. Wu and K.-M. Chao. *Spanning Trees and Optimization Problems*. Taylor & Francis Group, USA, 2004.

[143] X. Wu and G. Liu. XML twig pattern matching using version tree. *Data & Knowledge Engineering*, 64(3):580599, 2008.

[144] L. Xu. *Source Discovery and Schema Mapping for Data Integration.* PhD thesis, Brigham Young University, 2003.

[145] L. Xu and D. W. Embley. A composite approach to automating direct and indirect schema mappings. *Information Systems*, 31(8):697–732, 2006.

[146] I. Zaihrayeu. *Towards Peer-to-Peer Information Management Systems*. PhD thesis, University of Trento, 2006.

[147] Z. Zhang, H. Che, P. Shi, Y. Sun, and J. Gu. Formulation schema matching problem for combinatorial optimization problem. *Interoperability in Business Information systems (IBIS)*, 1(1):33–60, 2006.

[148] H. Zhao and S. Ram. Clustering schema elements for semantic integration of heterogeneous data sources. *Journal of Database Management*, 15(4):88–106, 2004.

[149] Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning 55(3): 311-331 (2004)*, 55(3):311–331, 2004.

[150] Y. Zhao and G. Karypis. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 10(2):141–168, 2005.

[151] E. Zitzler and L. Thiele. Multiobjective evolutionaty algorithms: A comparative case study and the strength pareto approach. *IEEE Tran. on EC*, 3:257–271, 1999.