UNIVERSITY OF CALIFORNIA, SAN DIEGO

Schema Mapping for Data Transformation and Integration

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in
Computer Science

by

Guilian Wang

Committee in charge:

Professor Joseph Goguen, Chair
Professor Pao Chau
Professor Chung-Kuan Cheng
Professor Yannis Papakonstantinou
Professor Victor Vianu

2006

The dissertation of Guilian Wang is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2006

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

It is difficult to overstate my gratitude to my PhD advisor, Prof. Joseph Goguen. I could not have imagined having a better advisor for my PhD. With his enthusiasm and intuition on practical problems in data transformation and integration, and his great efforts to explain things clearly and simply, he helped me in making this topic fun for me. Throughout my thesis research and writing period, he provided encouragement, sound advices, many ideas, patient teaching and warm caring. He also made careful arrangements to secure funding for my whole PhD program. Though currently he has to fight extremely hard against disease evil, he is still working as hard as usual on helping me with thesis writing. The first draft of this thesis was reviewed by him with pipes, wires and machines all around his body on the hospital bed. It is to his 65th birthday celebration that I dedicate this thesis.

I would like to thank Dr. Arcot Rajasekar for strong support and enthusiastic supervision for my final stages of thesis research in the NSF project SEEK. I also thank him for thoughtful advices and kind help on my career development. I thank for Dr. Tony Fountain for providing me nice support and environment for my first two years of study and continuous moral support for finishing the entire program. I appreciate Dr. Bertram Ludaescher, Dr. Shawn Bowers and Dr. Matt Jones for valuable discussions, encouragement and help on integration of our schema mapping tool SCIA with the workflow system Kepler. I thank Dr. Deana Pennington for her interests in my research, sharing data and integration problems in the Long-Term Ecological Research Program, and her friendship. I appreciate Prof. Young-Kwang Nam for long-lasting collaboration. I am very grateful to Dr. Kai Lin for collaborating on research and providing patient mentoring for issues at various stages of my study. I thank for Vitaliy Zavelov for working with me together on the SCIA user interface and experiments. Dr. Rami Rifaieh is appreciated for his interests in SCIA, ideas on applications and help with experiments.

Dr. Mark Miller provided financial support for conducting the formal experiments, here I appreciate him for that and his efforts on extending SCIA for more biological applications.

I wish to thank all my thesis committee members. In particular, Prof. Pao Chau helped me improve project proposals for graduate fellowship applications and explore chemical and biological data integration problems since my very first quarter at UCSD. I also appreciate his warm caring and strong support for my career development.

I am indebted to all faculty members and student colleagues in both Meaning and Computation Lab and Databases Lab for providing a stimulating and fun environment for me to learn and grow. They include Prof. Alin Deutsch, Prof. Yannis Papakonstantinou, Prof. Victor Vianu, Dipti Borkar, Emiran Curtmola, Dana Dahlstrom, Ryoko Goguen, Fox Harrell, Heasoo Hwang, Yannis Katsis, Monica Marcus, Kian Win Ong, Nicola Onose, Liying Sui, Avinash Vyas, Yu Xu and Dayou Zhou. Many of them volunteered for the SCIA experiments, here again I thank them and other fellows who participated in the experiments. I am grateful to the staff in our department for helping the department run smoothly and for assisting me in many different ways. Julie Conner and Steve Hopper deserve special mention. I wish to thank my friends Sheau-Yen Chen, Neil Cotofana, Sashka Davis, Longjiang Ding, Judy Ellies, Hongmei Guo, Jingfei Guo, Qing Liu and her family, Peter Shin, Martha Stacklin, Fang Wen, Jingjing Wu, Qiao Xin, Huizhu Xuan, Xianan Zhang, Xiaoping Zhang and many others for helping me get through the difficult times, and for all the emotional support, entertainment and caring they provided.

I cannot end without thanking my family, on whose constant encouragement and love I have relied throughout my time. I owe special recognition to my mother Ms. Dianhua Cao, who was always working hard and taking care of others throughout her short but lengendary life. I thank my father Mr. Weiru Wang for always believing in me and giving me and my brothers Gongfa Wang and Fei

Wang the best education he could. I also appreciate the examples of my grandma Ms. Chengying Han, grandpa Mr. Zhaoan Wang, aunt Ms. Qiuyue Wang, uncle Mr. Heming Huang, mother-in-law Ms. Chuanhua Shi, father-in-law Mr. Kegui Zhou and my husband Mr. Zhou Zhou. Their unflinching courage always inspires me and hopefully also our children David Zhou and Richard Zhou.

VITA

1993	B.E., Environmental Engineering, Tsinghua University, China
1996	M.S., Environmental Chemistry, Chinese Academy of Sciences (CAS)
1993–1996	Research Assistant, Research Center for Eco-Environmental Sciences (RCEES), CAS
1996–1997	Research Associate, Sino-Canadian High-tech Center of Resources and Environment (SCORE), RCEES, CAS
1997–2000	Assistant Research Professor, SCORE, RCEES, CAS
2000–2006	Research Assistant, Department of Computer Science and Engineering & San Diego Supercomputer Center, University of California, San Diego
2006	Ph.D., Computer Science, University of California, San Diego

PUBLICATIONS

Guija Choe, Young-Kwang Nam, Joseph Goguen and Guilian Wang. Information Retrieval from Distributed Semistructured Documents Using Metadata. in Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2006.

Guilian Wang, Joseph Goguen, Arcot Rajasekar, Young-Kwang Nam, Rami Rifaieh and Vitaliy Zavesov. Schema Mapping for Data Transformation and Integration. Final version in preparation for submission.

Joseph Goguen, Guilian Wang, Vitaliy Zavesov, Young-Kwang Nam. Schema Mapping Interface Design Principles. Final version in preparation for submission.

Guilian Wang, Joseph Goguen, Young-Kwang Nam and Kai Lin. Critical Points for Interactive Schema Matching. In Advanced Web Technologies and Applications, edited by Jeffrey Xu Yu, Xuemin Lin, Hongjun Lu and YanChun Zhang, Springer Lecture Notes in Computer Science, volume 3007, pp.654-664, 2004. Proceedings of the Sixth Asia Pacific Web Conference, Hangzhou, China, April 2004.

Joseph Goguen, Guilian Wang, Young-Kwang Nam and Kai Lin. Abstract Schema Morphisms and Schema Mapping Generation. Technical Report. Department of Computer Science and Engineering, UCSD, 2004.

Guilian Wang and Joseph Goguen. Analysis for Schema Matching Tool User Interface Design. Technical Report. Department of Computer Science and Engineering, UCSD, 2004.

Young-Kwang Nam, Joseph Goguen and Guilian Wang. A Metadata Tool for Retrieval from Heterogeneous Distributed XML Documents. in Proceedings of International Conference on Computational Science, Springer, Lecture Notes in Computer Science, Volume 2660, pp. 1020-1029, 2003.

Young-Kwang Nam, Joseph Goguen and Guilian Wang. A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases. in Proceedings of International Conference on Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems, Springer, Lecture Notes in Computer Science, Volume 2519, pp.1332-1344, 2002.

Tony Fountain et al. Demo of the Spatial Data Workbench: A System for Managing and Mining Large Geospatial Collections for Ecology. in SC2002: High Performance Networking and Computing. Baltimore, November 16-22, 2002.

Deana Pennington, Tony Fountain, Guilian Wang and John Vande Castle. Spatiotemporal Data Mining of Remotely Sensed Imagery for Ecology. in Proceedings of GIScience 2002, Second International Conference on Geographic Information Science, Boulder, Colorado, September 25-28, 2002.

Lu Yonglong, Liu Zongchao and Wang Guilian (Co-ed.). Strategic frame for sustainable development of China. Reformation Press, Beijing, 1999.

Lu Yonglong, Wang Guilian and Shi Yajuan. Prospective in development of environmental technology in China. Journal of Environmental Sciences. Vol.11, No.3, pp.303-308,1999.

Wang Guilian and Lu Yonglong. Integrated assessment of urban environment in China. Journal of Environmental Sciences. Vol.11, No.3, pp.272-278, 1999.

Wang Guilian and Bai Naibin. Comprehensive QSAR modeling system for typical contaminants. in New Trends in Chemometrics . First International Conference on Chemometrics in China , Zhangjiajie, October, 1997. Hunan University Press.

Wang Guilian and Bai Naibin. Structure-activity relationships for rat and mouse LD50 of miscellaneous alchohols. Chemosphere. Vol.36, No.7, pp.1475-1483,1998.

Xu Jian, Lu Yonglong and Wang Guilian. GIS/ES technologies applied to environmental emergency response systems. ACTA SCIENTIAE CIRCUMSTANTIAE. Vol.19, No.2, pp.190-194, 1999.(in Chinese)

Wang Guilian, Lu Yonglong and Xu Jian. Application of GIS technology in Chemical emergency response. Journal of Environmental Sciences. Vol.11, No.4, pp.272-278, 1999.

Wang Guilian and Bai Naibin. Study on QSAR for general pollutants in organic industrial waste . Toxic Substance Mechanisms . Vol.16, No.4, 1997.

(Co-trans.) Environmental management (Textbook for graduate students with major of environmental management in European Union). High Education Press, Beijing, 1996. (in Chinese)

Wang Guilian and Bai Naibin. Study on QSAR for chlorophenols . Fresenius Environmental Bulletin . Vol.5, pp.67-72, 1996.

Wang Guilian and Bai Naibin. QSAR Models for Chlorophenols. ACTA SCIENTIAE CIRCUMSTANTIAE. Vol.16, No.2, pp.190-194, 1996. (in Chinese)

Bai Naibin and Wang Guilian. An Expert System for Response to Chemical Emergency . Proceedings of Advanced Technologies for Resources and Environmental Enhancement, 1st SCH-CORE Workshop. April 24-28, 1995. Beijing, China.(in Chinese and English)

Wang Guilian and Bai Naibin. A Review of Quantitative Structure-Activity Relationship Models for Environmental Pollutants. Advances in Environmental Science. Vol.3, No.4, pp.39-45, 1995. (in Chinese)

FIELDS OF STUDY

Major Field: Science

Studies in Computer Science.

Professor Joseph Goguen, University of California, San Diego

Major Field: Science

Studies in Environmental Chemistry.

Professor Naibin Bai, Research Center for Eco-Environmental Sciences, Chinese Academy of Sciences

Major Field: Engineering

Studies in Environmental Engineering.

Professor Deli Xi and Kebin He, Tsinghua University

ABSTRACT OF THE DISSERTATION

Schema Mapping for Data Transformation and Integration

by

Guilian Wang
Doctor of Philosophy in Computer Science
University of California, San Diego, 2006
Professor Joseph Goguen, Chair

Finding semantically correct mappings between the schemas of two data sources is a fundamental problem in many important database applications. But it cannot be fully automated, so user input is necessary. This dissertation presents SCIA, a system that assists users in creating executable mappings between a source schema and a target schema by automating simple matches and finding critical points where user input is necessary and maximally useful. SCIA handles complex mappings involving n-to-m matches with semantic functions and conditions, which often exist in practice, but are ignored in most related systems. It outputs mappings in both correspondence format and executable view format that can transform source data into target instances. Formal experiments show SCIA significantly reduces total user effort by using path contexts and combination of multiple matching algorithms to find critical points, and help users at those points by asking them specific questions with sufficient context and suggestions for adding semantic information for data transformation, such as join and grouping conditions. Moreover, SCIA has semantic models for schemas and schema mappings using algebra. This dissertation also gives principles for schema mapping user interface design, based on the ideas of minimal model for schema and schema mapping, and of optimal semiotic morphism from this model into the interface design.

Chapter I

Introduction

This dissertation studies the problem of schema mapping. Schemas are structural representations of data. Schemas considered in this dissertation include DTDs, XML Schemas, relational schemas and ontologies. The goal of schema mapping is the creation of semantic mappings between two schemas.

It takes two steps to create semantic mappings between a source schema and a target schema, as illustrated in Figure I.1. The first step is *schema matching*, to create a set of correspondences among schema elements, called schema matches. For example, in Figure I.1, schema matches say that `book element` in the source schema corresponds to `publication` element in the target schema, and that `fname` and `lname` in the source schema are related to `name` in the target schema. Notice that these correspondences only say conceptually the two elements mean the same thing in the real world, but they don't carry semantic information for data transformation. For example, it does not say how to combine `fname` and `lname` into full `name`. The second step is *view generation* or *query discovery*. View generation includes two steps: first, to add semantic information for data transformation on top of those correspondences, including join and grouping conditions, other filtering conditions and conversion functions, taking into account the structures of schemas, constraints and application-specific requirements; and then to generate an executable transformation code in format of a query language or

a general programming language. At the concept level, semantic mappings refer to correspondences or matches in schema matching tools. However at the data level, to schema mapping tools, semantics mappings refer to mapping expressions or transformation code that carry semantic information for data transformation.

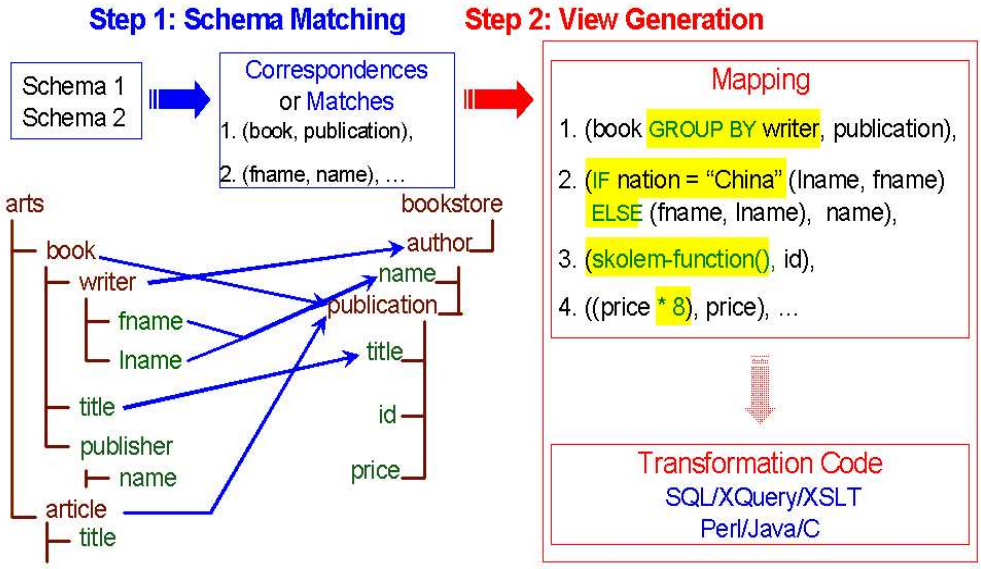

Figure I.1 Schema mapping steps

I.A Schema Mapping Applications and Context of the Dissertation

Schema mapping is a crtical step in many important database applications that require manipulating schemas or their underlying data. The most important applications are virtual data integration and data transformation in which semantic mappings both at concept level and the data level are required. Virtual data integration [66, 51, 29, 53, 52, 92] builds a uniform query interface to multiple, often heterogeneous, data sources, through a uniform query interface, allowing users to query over a global schema. The essential problem is to provide semantic mappings between the global schema and local schemas in order to reformulate user queries into appropriate queries to the data sources. A natural extension of virtual data

integration is peer data management which allows peers i.e., participating data sources, to query directly from each other; this also requires semantic mappings among peers for answering peer queries [28]. Data transformation is critical from one service to another in e-business and scientific workflows, and from one schema to another in data warehousing [64, 82, 12]. Semantic mappings are also used to guide the generation and optimization of transformation code.

As a matter of fact, this dissertation research is motivated by my own experience in environmental information applications [97, 93, 94, 21, 79]. A typical task in such information application includes accessing and integrating relevant data, analyzing the data and reporting the results. These can be captured as workflows. It is desirable to develop a workflow system that allows users to easily design such workflows and execute them efficiently. I have been participated in the development of an open source workflow system, Kepler, for the large collaborative NSF project Scientific Environment for Ecological Knowledge [26]. Our collaborators from ecology and biology, who are also the users, desire flexible and convenient data integration, data and analytical tools integration, and transformation between workflow components, in order to simplify workflow design for their dynamic analyses.

Besides data transformation and integration, the targeted applications of schema mapping in this dissertation, there are many metadata management applications in which schema mapping plays the key role, such as schema integration and model management. Schema integration is to merge a set of input schemas into a new global schema that captures everything in the input schemas. The merging process relies on semantic mapping among input schemas [5, 77, 83]. In model management, which aims at building tools for easily manipulating models of data [83, 2, 9, 63], merging schemas is also an essensial operation, and computing the difference between two schemas also requires first establishing semantic mappings between them. All the above issues arise in numerous traditional and emerging domains such as bio/eco/enviroinformatics, ubiquitous computing and

e-commerce. They are permeating into all areas of our daily life as the Internet brings its explosive growth of information online, and as the idea of Semantic Web is put into practice.

I.B Schema Mapping Challenges

Schema mapping is still a very challenging problem in general, though it has been studied actively for several years. The fundamental difficulty is to infer real-world semantics of data from the syntactical clues in their representations or their values, because these clues, including schema element names, types, local structures, constraints and value patterns, are often ambiguous, even misleading, and hence unreliable. For example, the same data at different sources may use different data models and representations, structural conventions and naming conventions. Let's look at two real schemas from bibliograpgy in Figure I.2. Different names author and writer are used for elements that share the same meaning in the real world. Bookstore schema uses fullname, however arts schema uses seperated firstname and last name. Even for the elements that do refer to the same concept, but they may still be non-equivalent due to differences in units, resolution, precision and measurement protocols. These aspects of information are often hard to interpret, and difficult to acquire; often they are undocumented or unkown. For example, arts books may be priceless, even there exist prices, but may be in Canadian dollars, while bookstore uses USA dollars. Since both id and price are required, it has to be specified how to get values for them through Skolem functions, manually provided, or obtain from another database through a join.

Moreover, schema mapping depends on the specific application. For example, a specific application determines whether to map arts article to publication in bookstore, and whether to group books from arts by writer, and group by which attributes of writer, and the order of concatenation of first name and last name for full name. These reasons make the full automation of schema mapping im-

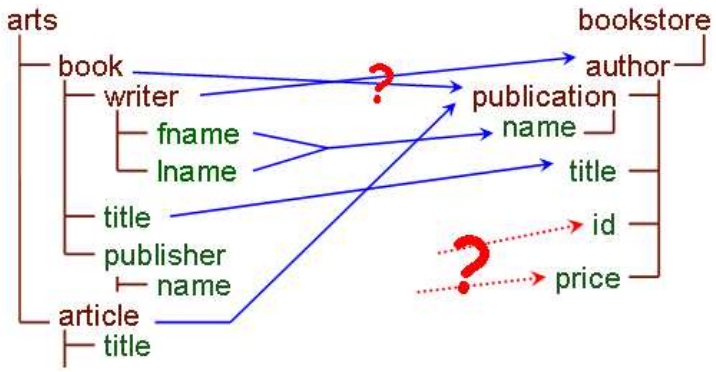

Figure I.2 Challenges in mapping bibliographical DTDs

possible. To create correct mappings between non-trivial schemas, human users familiar with all the schemas, their underlying data, and the application that uses the mappings must be involved.

However, it is extremely labor intensive and error prone for humans to manually create schema mappings, especially to write complicated queries or view definitions all by hand. Even worse, the users of schema mappings are often domain scientists, instead of computer experts.

I.C State of the Art

There are many efforts to develop methods for automating the schema mapping process. They mainly aim at finding interschema correspondences, i.e., semantic mappings at the concept level, which is usually called schema matching. An extensive review of techniques and tools for automatic schema matching up to 2001 is given in [82]. For two input schemas, $\mathcal{S}$ and $\mathcal{S}'$, most schema matching approaches such as instance-and-learner-based Semint [55, 56], LSD [19], GLUE [18] and MGS [48], and rule-and-schema-based Cupid [59], SF [62], Rondo [63], COMA [17] and COMA++ [4] only help find the most common case, i.e., 1-to-1 matches, in the format of (e, e') or (e, e', v) where e is an element from $\mathcal{S}$, e' is an element from $\mathcal{S}'$, v is a similarity value usually in the range of [0, 1], with a higher

v indicating that e and e' are more similar. Each individual matching technique has its own strength and weakness, hence more and more matching tools are being assembled from multiple components [17, 20, 86]. Recently more effort is on the engineering issues of schema matching, how to combine multiple components, such as COMA++ [4] and Protoplasm [10], and how to tune a schema matching system for a domain, like eTuner [86]. All such tools have great difficulty with matches that involve conditions or conversion functions, and cannot automatically discover n-to-m matches for $n > 1$ or $m > 1$. Instance-based COMAP [18] finds complex matches based on user input and real data (including target instances that are not available as input in the context of data transformation or virtual data integration). Some other tools [99, 11] use ontologies to discover some indirect matches that involve composition or decomposition of multiple elements based on user-provided ontologies.

Generating queries or view definitions from the match results for data transformation and query reformulation, i.e., semantic mappings at the data level, is called view generation. Semi-automatic tools with GUI have been developed, such as Clio [46, 64, 81, 80, 47], MapForce [3] and Stylus Studio [90]. However these tools assume that interschema correspondences are given, so that finding complex mappings, a significant proportion in practice, still remains a challenging problem.

In practice, semantic mappings are still created manually, though a few tools above claim with high accuracy and broad applicability. Because it is very difficult for users to identify and correct wrong and missed matches in output mappings without system help once the mapping process is done in those tools, their practical use is limited. The slow and expensive manual schema mapping has now become a serious bottleneck in various applications. With the fast growth of the Internet, Semantic Web, cross-disciplinary domains and e-commerce, semantic mappings will be needed at larger and larger scales, e.g., among millions of sources. Manual mapping is clearly impossible at such scales. Hence the development of

schema mapping solutions is more and more crucial in many data sharing applications. This dissertation studies the problem of schema mapping and develops semi-automatic tools that can greatly reduce total user effort.

I.D Goals of the Dissertation

The objective of this dissertation is to develop an approach to schema mapping, particularly applicable to data transformation and integration. The specific goals are:

1. Get first-hand experience on the problem of schema mapping through building data integration systems to better understand the challenges, requirements, total user effort, useful metrics, tradeoffs and semantics.
2. Develop an approach to schema mapping that can reduce total user effort.
3. Give principles for designing convenient schema mapping user interfaces.

To achieve these goals, we undertook with the following steps:

1. Build a data integration system for distributed heterogeneous XML data (Chapter II).
2. Develop an approach for iteratively finding not only simple mappings automatically, but also critical points of a mapping process, and assisting users identify complex mappings by using combination of multiple matching algorithms exploiting different aspects of schema information and path contexts at those critical points. Design a convenient GUI for easy interaction between users and the implemented system. Evaluate the solution by applying it to the real world data in multiple domains and to workflow design and composition in the workflow system Kepler in the SEEK project [26] (Chapter III).

3. Analyze the user interfaces of available schema mapping systems based on the principles and methods of algebraic semiotics. Give principles for schema mapping user interface design and apply them to the design of SCIA (Chapter IV).

I.E Contributions

The major contribution of this dissertation is an approach that bridges the gap between manual schema mapping in practice due to the shortage of satisfying solutions and recent automatic mapping techniques that ignore complex mappings or require expensive or unavailable information for finding complex mappings. It advocates the idea of *critical points* for schema mapping where user input is necessary and maximally useful, and saves total user effort by helping users at such points. It also provides a novel way to combine multiple techniques, each exploiting a certain type of information, through context check to find critical points and create mappings. Up to now, it is the only approach that supports both interschema correspondence discovery (i.e., schema matching) and query discovery (i.e., view generation); this is highly desirable for data transformation and integration. It can handle XML DTD and Schema, relational schemas and ontologies in the matching step; it can handle XML in the view generation step now. It has potential for handling any kind of schema because of its underlying semantics [34, 40].

In addition, this dissertation contributes to the area of user interface design. It introduces schema mapping as an important application of algebraic semiotics. It constructs the minimal models of familiar schema mapping tools, i.e., their source semiotic spaces; analyzes the semiotic morphisms from those source spaces to the user interfaces of the tools; provides suggestions for improvement; and propose design principles for schema mapping user interfaces. This will give a general model for obtaining user interface design principles for other application areas.

I.F Organization

There are 3 chapters following this introduction, each for one of the steps discussed in section I.D to achieve the goals of the dissertation. Another chapter discusses related work, and the last chapter summarizes the dissertation and talks about directions for future work.

Chapter II

Data Integration via Manual Schema Mapping

This chapter describes our experience with schema mapping in our joint development of the DDXMI (Distributed Document XML Metadata Interchange) data integration system with Prof. Nam's lab in Yongsei University in Korea [66, 67, 14, 15].

To get experience with the problem of schema mapping and to better understand the challenges, requirements, total manual effort, useful metrics, tradeoffs and semantics, we began by trying to build a data integration system that allows users to query against a global schema in a uniform query interface for a set of distributed heterogeneous local XML data.

We chose data integration because it is one of the most important data management applications today and for the future, and it requires semantic mappings at the lowest level. Hence it provides a general problem setting for understanding schema mapping, the solution of which could be directly applied to data transformation, and it provides more than enough for metadata management applications that require semantic mappings only at the concept level.

Because the wrapper-mediator approach for data integration is mature and popular, and the XML data model is becoming more and more widely used, as

its query languages and implementations become available, we decided to focus on integrating XML data sources first, by using semantic mappings between a global schema and a collection of local schemas. In this project, semantic mappings are assumed to be supplied by integration engineers who understand both the semantics of the application global schema and the semantics of all participating local schemas and their underlying data.

II.A Integration Problems and Requirements of Schema Mappings

To understand semantic mappings for answering queries against the global schema using participating local data sources, we first looked at real data in the domains of bibliography, biology, ecology and business. We chose bibliography because bibliographical data are used in the use cases of W3C XQuery, and there are many data sources for bibliography available online; furthermore, integration of bibliographical data is an important part in building information systems such as digital libraries and online book stores. The bibligraphical data sources in our experiment have schema information specified in by DTDs. We chose one of them as the global DTD. The domains of biology and ecology were chosen because the objective of the SEEK project we are woking in is to develop easy tools for biologists and ecologists to integrate and analyze their data. The biological data are about genetic sequences encoded in different popular markup languages, including BSML [49], GAME [54] and BIOML [57], ecological data are from field observations of biodiversity at various LTER (for Long Term Ecological Research) sites [1, 26]. The business data are about purchase order Schemas of 5 different companies in XML Schema format, used in Coma [17], originally from `www.biztalk.org`.

An element from one schema can participate in one or more matches, causing *global cardinality* of 1-to-1, 1-to-n, n-to-1 or n-to-m. Within a match, one or more elements from one schema may be matched with one or more elements of

another schema, causing *local cardinality* of 1-to-1, 1-to-n, n-to-1 or n-to-m [83]. Therefore cardinality depends on match representation. The following are some examples from the above integration problems.

Example 1: global 1-to-n in bibliography. `bookstore/book/author/name` from book3.dtd corresponds to both `books/book/author/full-name/last-name` and `books/book/author/full-name/first-name` in book.dtd. Grouping them together, we have local cardinality 1-to-2; considering obtaining data for each target element from the source, it is more natural to seperate them into two local 1-to-1 matches, so we have global cardinality 1-to-2. The functions *lstring* and *fstring* for extracting the last name and the first name from a full name are needed.

Example 2: local n-to-1 in bibliography. The converse direction of the above example is that the concatenation of `books/book/author/full-name/last-name` and `books/book/author/full-name/first-name` corresponds to `bookstore/book/author/name` in book3.dtd. If seperated into two 1-to-1 matches, they would not provide the correct mapping information for data transformation, because there is no information about how to combine the two source elements.

Example 3: local n-to-1 in ecology. In the ecological domain, species density equals species count divided by the area in which the species is counted. Some data sources record the count and area, others record only the computed density. Hence, both `studyA/obs/count` and `studyA/obs/area` in studyA.xsd from one LTER site match to `observations/observation/density` in the global schema for bio-diversity research. The function *div* is needed to combine the data of these two local elements into a single global element.

Example 4: global n-to-1 in biology. `BSML/Definitions/Sequences/Sequence/Segment` corresponds to `Sequences/Sequence/Segment`, and `BSML/Definitions/Sequences/Sequence/Feature-tables/Feature-table/Feature` corresponds to `Sequences/Sequence/Segment`. It differs from the above local n-to-1 matches because `Sequences/Sequence/Segment` is the union of `BSML/Definitions/Sequences`

/Sequence/Segment and BSML/Definitions/Sequences/Sequence/Feature-tables/Feature-table/Feature, if there are x Segment elements and y Feature elements under BSML/Definitions/Sequences/Sequence, then there will be $x+y$ Segment elements under a specific element Sequences/Sequence that comes from BSML/Definitions/Sequences/Sequence. To make match representation as simple as possible, it is desirable to seperate them into 2 local 1-to-1 matches, resulting in global cardinality 2-to-1. Hence, global many-to-1 matches are also called **union** matches.

Example 5: global n-to-1 in business. In the purchase order schema from Apertum APERTUM-PO/InvoiceTo/Contact as well as APERTUM-PO/DeliverTo/Contact in the purchase order schema from Apertum, correspond to Excel-PO/Header/Contact in the purchase order schema from Excel. Hence it is necessary to take the union of the first two for the third.

The use of different reference systems often causes conflicts that demand conversion functions even for 1-to-1 matches.

Example 6: complex 1-to-1 in bibliography. The price of books may use the American dollar in the global book.dtd, but in local book3.dtd may use Canandian currency, or be represented in cents.

Even more complicated, sometimes conditions have to be satisfied when creating a global element instance from its corresponding local element values.

Example 7: complex 1-to-1 in biology. The match of BSML/Definitions/Genomes/Genome/Organism to Sequences/Sequence/Organism is correct for data transformation only if the condition Sequences/Sequence/genomeRef = BSML/Definitions/Genomes/Genome@id holds, assuming that the same path is for the same element. The condition is to make sure that the Sequence is from the Orgnism through checking the Sequence belongs to a Genome of the Organism.

Example 8: complex local n-to-1 in bibliography. In the above Example 2, the order of concatinating author's last-name and first-name into full name may

depend on the `nationality` of the author. Such 1-to-1 matches with conditions are often more naturally represented as many-to-many matches.

After the most common 1-to-1 matches, complex ones like the above examples are significant for each of the above integration problems. The simple match representation (<src-elt>, <tgt-elt>) used in most matching algorithms is far from adequate for these complex cases. n-to-1 matches only makes sense when the way their elements relate can be specified, such as by functions or union types that are available in mappings. Considering the tradeoffs between requirements on both the simplicity of implementation and the expresiveness of match representation, we decided to separate union matches into simpler matches with union treated implicitly to distinguish natural many-to-1 matches, and we chose the following left-to-right EBNF (Extended Backus-Naur Form):

([<src-elt>,] [<fcn>] , <fcn>, <src-elt> [, <fcn>]* [, <cdn>], <tgt-elt>)

where <src-elt> is a source element, <fcn> is a function name, <cdn> is a condition expression, and <tgt-elt> is a target element. Please notice that we use element and path interchangely. For a target element, it allows a function for manipulating the content of a source element to its left, and a function for combining the result of its left side and the content of a second source element if there is no function for the second source element otherwise the output of a function over the second source element itself, and then another function for combining a third source element, and so on. Hence it allows combination of multiple source elements. If there is no corresponding <src-elt> in a mapping element, i.e., 0-to-1 match, the target value can be a constant or generated by a function, by <fcn> we cover constants for simplicity. The occurrence of multiple mapping elements for the same target element <tgt-elt> implicitly means the union of all of them makes the complete mapping element for that <tgt-elt>.

II.B Representation and Creation of Schema Mappings

Inspired by XMI [45], we used a structured document to store the semantic mappings, and called *DDXMI* (for Distributed Document XML Metadata Interchange)in our joint project with Prof. Nam's lab [66, 67, 14, 15]. The structure of DDXMI has been evolving as our work goes on. The latest version of the DDXMI DTD is as follows. Notice it does not support condition yet.

```
<!ELEMENT DDXMI (header, isequivalent)>
<!ELEMENAT header (documentation, version, date, authorization)>
<!ELEMENT documentation (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT authorization (#PCDATA)>
<!ELEMENT isequivalent (global, component*)*>
<!ELEMENT global (#PCDATA)>
<!ELEMENT component (match*)>
<!ELEMENT match (local, (op, local)*)>
<!ELEMENT local (#PCDATA)>
<!ELEMENT op CDATA>
<!ATTLIST local operation CDATA>
```

The elements in the global schema are labeled `global`, their corresponding elements in local schemas are labeled `local`. Each `component` is for a single data source, the union of all the `match` elements is equivalent to the corresponding `global` element. The `operation` attribute of `local` element contains the name of the function to be applied to the `local` element content, `op` contains the name of the function to combine one `local` element content with another.

To save user effort in specifying and recording mapping information in such a DDXMI document, a simple GUI is provided for integration engineers. First, the system assigns an index to every path in the global schema tree and displays it in the GUI. Then mapping is done by assigning the same index numbers to corresponding paths or clicking the corresponding paths across the schema trees

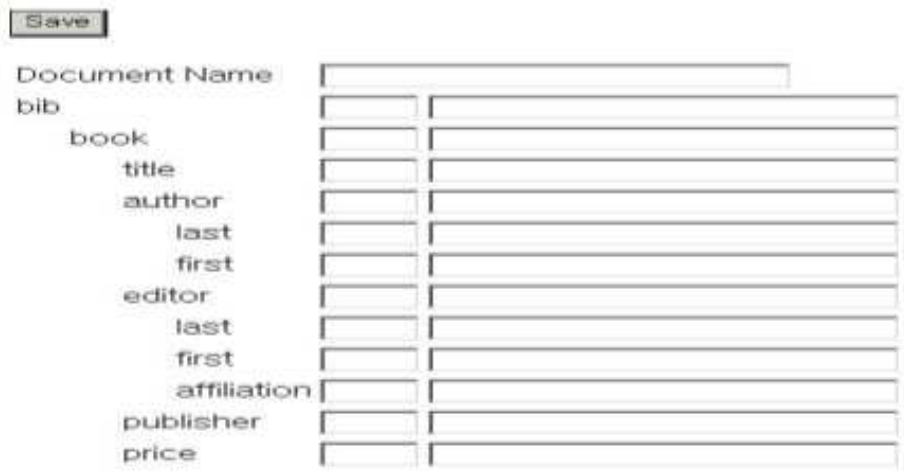

Figure II.1 The GUI for indexing schema elements

and adding function names. Figure II.1 shows the schema tree in GUI, with a field for assigning index and another field for specifying functions for each node. By collecting the paths and functions with the same number, the mappings for each global path are obtained and a DDXMI file is generated. For example, assume we want to build a query interface over three book information sources. The desirable target or master schema tree and the three local schema trees are shown in Figure II.2. The same index number is given to elements that have the same meaning, and each index sequence is different. Some nodes in the schema tree have no number, since the master index does not include it. Book1.DTD and Book3.DTD have a 'price' node but Book2.DTD doesn't. If some query includes the 'price' element, then no query would be generated for Book2.XML since Book2.DTD doesn't have a price element. Based on the index assignment in Figure II.2, the DDXMI file is generated automatically by collecting paths with the same numbers.

II.C Query Reformulation

To answer a query against the global schema, the system generates a set of queries to relevant local schemas by consulting the mapping information in the DDXMI document. To generate a local query for a specific data source, each global

[A Master Library document DTD Paths]

0	Book.xml
1	/book
11	/book/@price
12	/book/author
1211	/book/author/full_name
1212	/book/author/full_name/first_name
13	/book/author/full_name/last_name
14	/book/title
15	/book/year
16	/book/publisher
161	/book/editor
162	/book/editor/affiliation
162	/book/editor/full_name

[Book1.Index]

0	Book1.xml
1	/bib/book
11	../price
12	../author
1211	../author/first
1212	../author/last
13	../title
14	../@year
15	../publisher
16	../editor
161	../editor/affiliation
162	../editor/last
162	../editor/first

[Book2.Index]

0	Book2.xml
1	/arts/book
12	../author
1211	../author/firstname
1212	../author/lastname
13	../subject
14	../@year
15	../publisher

[Book3.Index]

0	Book3.xml
1	/bookstore/book
11	../price div(100)
12	../author
1211	../name fstring
1212	../name lstring
13	../title

Figure II.2 The indexed schema trees

path in the user quesry is replaced by the corresponding paths in this data source schema. If none of the global paths in the user query has any corresponding local paths for this data source, then there will be no local query for this data source.

II.D Implementation, Experiments and Results

Figure II.3 shows the architecture of the DDXMI system. It has been implemented for the NT operating system using a Java servlet server and a JavaCC compiler. Quilt is the XML query language of the prototype implementation. Kweelt, developed in University of Washington, is used as a query engine to execute quilt queries. DDXMI was tested with data from bibliographical domain [66, 67], and business data [14, 15]. Figure II.4 and II.5 show examples of Quilt queries with their generated local queries and the result of their executions.

II.E Discussion

However, for semistructured data, structural information may not be given explicitly, and data may be created without any restriction on structure.

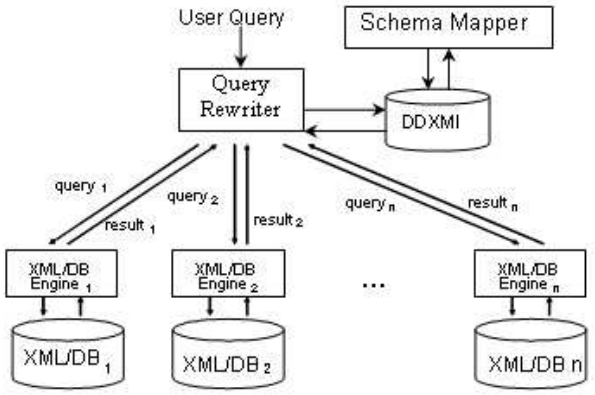

Figure II.3 The DDXMI system architecture

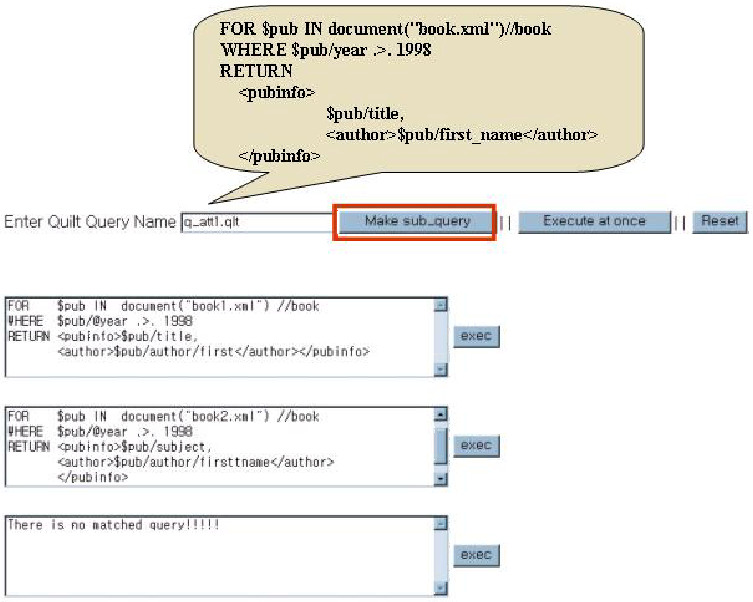

Figure II.4 An example of local query generation

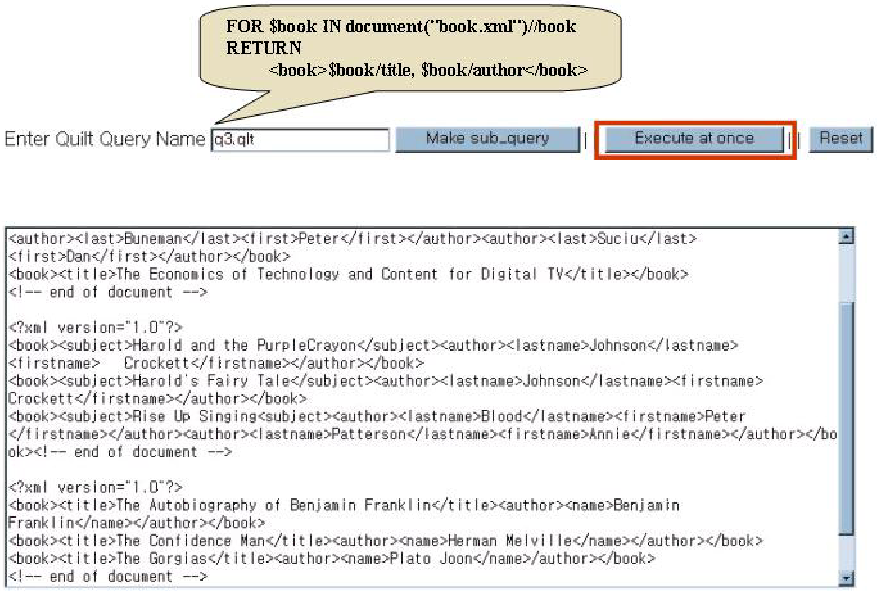

Figure II.5 A query execution result over 3 bibliogaphy documents

To provide a uniform query interface over them, schema extraction functionality is needed. To facilitate formulation, decomposition and optimization of queries for semistructured data, schema extraction or type inference have been studied by using machine learning methods and heuristics [69, 71, 70]. Unfortunately, the accuracy goes down as the extracted schema size decreases. In addition, it often happens that elements with the same tag have different structures and contain different information so that several elements with the same tag and even the same path in an extracted local schema correspond to a single element in the global schema, even worse, some of the elements with the same tag are really mapped, while the others are not and so they have to be filtered out in the local query. We proposed a node identification mechanism based on substructures and an algorithm for generating predicates to filter out the unwanted elements with the same tag as wanted ones to solve this problem in [14].

II.F Summary

We studied schema mapping by developing a query interface over multiple data sources. The schema tree of data sources are visualized from their associated schemas or the documents themselves if there is no given schema information, allowing a data integrator to match nodes with the same meaning by just clicking the node or assign index numbers and to name any necessary semantic functions for resolving representation differences. Then the DDXMI file is generated by collecting the local paths mapped to the global node. Global queries from end users are translated to appropriate queries to local documents by looking up the corresponding paths and possible semantic functions in the DDXMI document, and node identification information. Finally local queries are executed by Kweelt.

There are several obvious limitations to the query processing algorithm and its implementation. First, we extract the tree of nodes for documents without explicit schemas using an algorithm that may produce extremely large trees of nodes for irregular semistructured data, which may be too difficult for human to handle. It is desirable to explore how to balance the accuracy and size of approximate typing in practice. Secondly, JOINs among local data are not considered. Furthermore, the current mapping mechanism only allows describing is-equivalent mappings between the global paths and local paths. In order to fully use knowledge of the local documents for query decomposition and optimization, it is desirable to support describing and using relationships at more levels, such as local path vs. local path, document vs. document, and document vs. path. Unfortunately the simple path substitution mechanism used in query reformulation doesn't guarantee the returned query results conform to the global schema. Finally, conditional matches are not supported in the current DDXMI implementation.

One of the lessons learnt from this project is that simple schema matches do not provide sufficient mapping information for rewriting global queries into appropriate local queries, or for integrating the local query results into a consistent

global schema. As a result, the query result from the current DDXMI prototype is not guaranteed to be consistent with the global schema. Data integration requires more fine-grained semantic mappings, e.g., queries that define the global schema as a view over each local schema and that can be executed immediately to translate local data to a global schema.

In addition, our experience with DDXMI confirms that it is desirable to assist users by partially automating the process of schema mapping (including both the schema matching part and the view creation part), although the complexity of the problem is such that full automation is impossible in general.

Chapter III

Critical Points for Schema Mapping

It is widely recognized that it is impossible to fully automate the schema mapping process with complex matches that involve multiple elements, functions and/or conditions. Some tools rely on user input of domain knowledge, even a formal ontology, to find complex matches, and they often require much manual effort on post-match editing. Typically, the user has to convert the matching results to a format understandable by a view generation tool. Some information may be lost and must be input again manually. It also takes time to select the correct view definition among all generated possible ones. We think that schema matching and view generation are interdependent, so it is desirable to have a tool that handles both together.

From our SEEK project [26] and other similar NSF ITR projects addressing scientific data integration and analysis for specific domains, we have found that it is extremely painful for domain scientists to formulate their knowledge into systematic ontologies. On the other hand, it is easy for them to provide feedback on how two specific elements in specific schemas for familiar data sources relate to each other. Considering this reality, we choose to acquire semantics through interactions with users, instead of relying on predefined ontologies; however we

can take advantage of ontologies if they are available.

To save the total user effort, we introduce the idea of **critical points** for interactive schema mapping, and implemented it in our schema mapping tool SCIA [96, 40]. This chapter is organized as follows: we discuss the idea of critical points in Section III.A; we describe how to detect and resolve critical points in Section III.B; then we give an overview of SCIA in Section III.C, and describe its applications in biodiversity research in Section III.D and its applications in workflow system Kepler in Section III.E; finally we report the qualitative and quantitative evaluations in Section III.F and III.G. Some materials here are based on publications and ongoing paper drafts coauthored with Goguen, Lin, Nam and Zavelov [96, 40, 41].

III.A Critical Points for Schema Mapping

A **critical point** in a schema mapping process occurs when a core context has either no good matches, or else has more than one good 1-to-1 match, where **core contexts** are the most important contextualizing elements for tags within subtrees. Since tag meanings vary with context, and context is given by higher elements, correctness of core context matches greatly affects correctness of matches for all nodes under them. Core context elements typically have a large subtree, and can be found by heuristics and/or user input. The following are some examples of critical points:

(1) When multiple almost equally good match candidates are found, global multiple correspondences may exist; e.g., in Example 5 in Section II.A, purchase order header `Contact` might match `Contact` for billing, shipping and/or supplier, and only the user can say which is right, or if there are others.

(2) When multiple similar but poor match candidates are found, local multiple matches may exist and semantic functions and/or conditions may be involved; e.g., in Example 2 in Section II.A, `first-name` and `last-name` for `name`,

and in Example 3 in Section II.A a species `count` divided by `area` of the studied field for that species `density`.

(3) When possibly corresponding paths have inconsistent contexts, e.g., if `Bib/Book/Author` is found as best match candidate to `arts/article/author`, or if `BSML/Definitions/Genomes/Genome/Organism` is found the best candidate to `Sequences/Sequence/Organism`, then users should be asked.

(4) When no good match is found, users can say if a match exists or is indirect; if no match exists, then dealing with its subtree is also an issue.

III.B Detecting and Resolving Critical Points

Challenges exist for identifying critical points and providing significant help to users at critical points. We determine critical points using path contexts and a combination of multiple matching algorithms exploiting different aspects of information in schemas. We call the process of detecting and resolving critical points as *context check*. Context check does both matching and mapping. It computes new matches based on path contexts; adds semantic information for data transformation, i.e., turning simple matches into mapping elements.

Using path contexts is helpful for mapping complicated XML hierarchical structures. For example, in the above example (3) for critical points, the context checker will suspect the correctness of the match of `Bib/Book/Author` to `arts/article/author`, and also that of the match of `BSML/Definitions/Genomes/Genome/Organism` to `Sequences/Sequence/Organism`, and hence ask users for feedback. When provided with helpful information including the doubt about the consistency of contexts of the best match candidates and the proposed condition, users can easily figure out where to work, and then deny the first match and confirm or edit the proposed condition for the second. Denying the first match avoids all mistakes made on the subtrees. The details of the context checker are given in section III.C.5.

It is recognized that combining multiple matching algorithms to exploit multiple types of information can improve matching accuracy [17, 19, 59, 18, 86, 4, 10]. Hence we also use multiple different matchers, e.g., name matcher, path matcher, type matcher, description matcher, structural matcher, context matcher. We combine them in a flexible way to find a good match for each target element. When a good match for a core element cannot be determined by combining the results from all matchers, and when it is hard to determine how to combine the results from different matchers, i.e., at critical points, we ask users specific questions (with useful background information).

There are also challenges on effectively using structural information in schemas and combining structural matchers with other matchers. The intuition underlying algorithms for structural matching is that neighbors of similar nodes are similar. For example, SF (Similarity Flooding) is implemented by propagating similarity values to neighbors iteratively until convergence is achieved [62]. When combined with other matchers that may provide good results for some points already, however, the unselective propagation among neighbors may make the difference smaller. To solve this problem, we designed a selective propagation that freezes matches of very high similarity from previous steps of propagation steps supplied by other matchers and users.

III.C Implementation of SCIA

We developed SCIA, based on the semantics of schema mapping in [34, 40] and also the idea of critical points [96]. SCIA finds critical points, does as much as reasonable automatically, identifies new critical points, and iterates these steps until convergence is achieved. Figure III.1 depicts mapping for one source and one target schema. SCIA has an automatic mode that does just one pass using default strategies, but interactive mode may do several iterations, with user choices of candidates and matching strategy at critical points. Each iteration has four steps

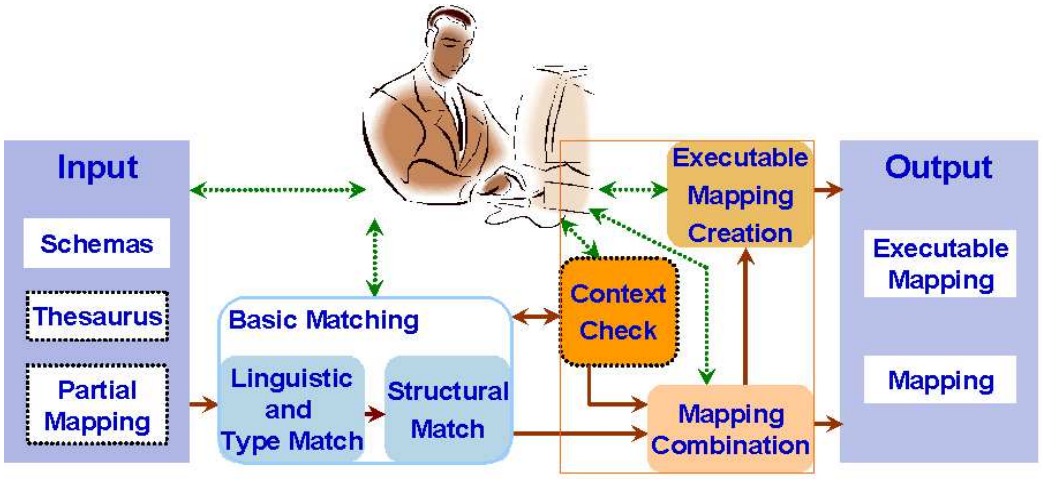

Figure III.1 Overview of the SCIA mapping process

for matching: linguistic and data type matching, structural matching, (optional) context check, and combination of match results; all matching steps and the view generation step have optional user input. Linguistic matching utilizes schema element names, paths and textual description. SCIA does not require pre-match effort, though user input of matches, domain thesauri (or ontologies) and input of previous matches at the beginning can greatly help.

SCIA creates mappings in both path/concept correspondence format and executable view format to support transformation of data from the source schema to target schema and answer queries over target schema. The path correspondences are the input to the view generation step. They can be saved in a document with structure specified in the XML Schema file in Appendix C, and can also be reused in SCIA for future mapping tasks, or as input to another view generation tool. Currently, SCIA outputs view definitions as Quilt expressions and uses Kweelt engine to execute the generated Quilt Queries for verifying data translation.

Figure III.2 shows the automatic matching results before context check between books-book.dtd, which groups writers by book, to books-author.dtd, which groups books by author. Some real matches have been recognized, for example, `writer` to `author`, book's `title` to book's `title`. Some predicted matches are

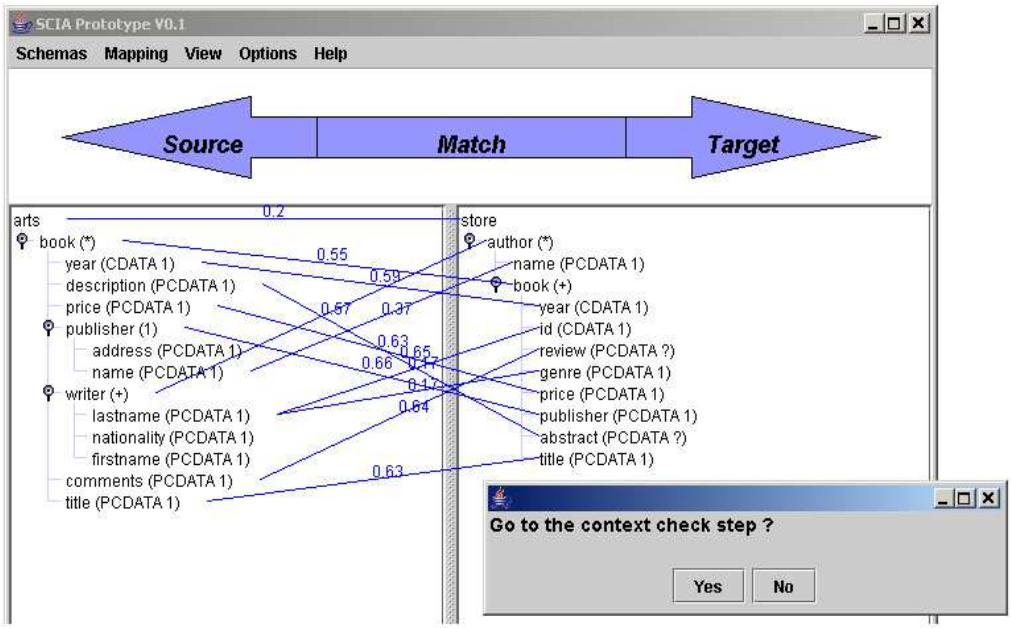

Figure III.2 Automated matching results

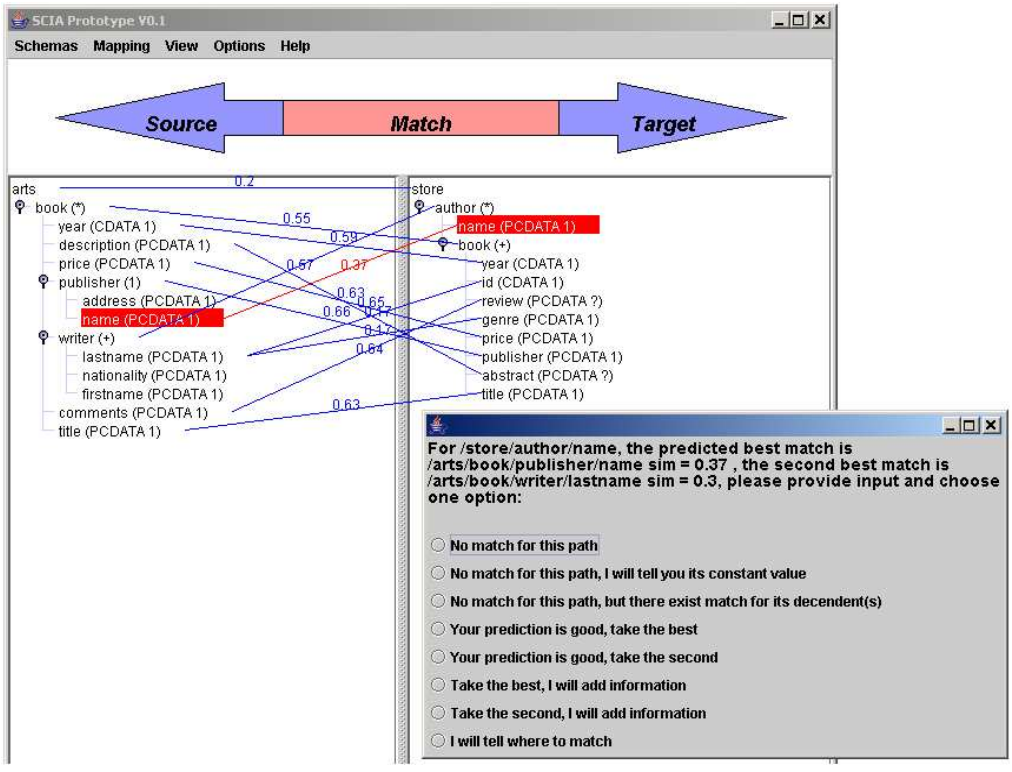

Figure III.3 A crtical point in mapping bibilographical DTDs

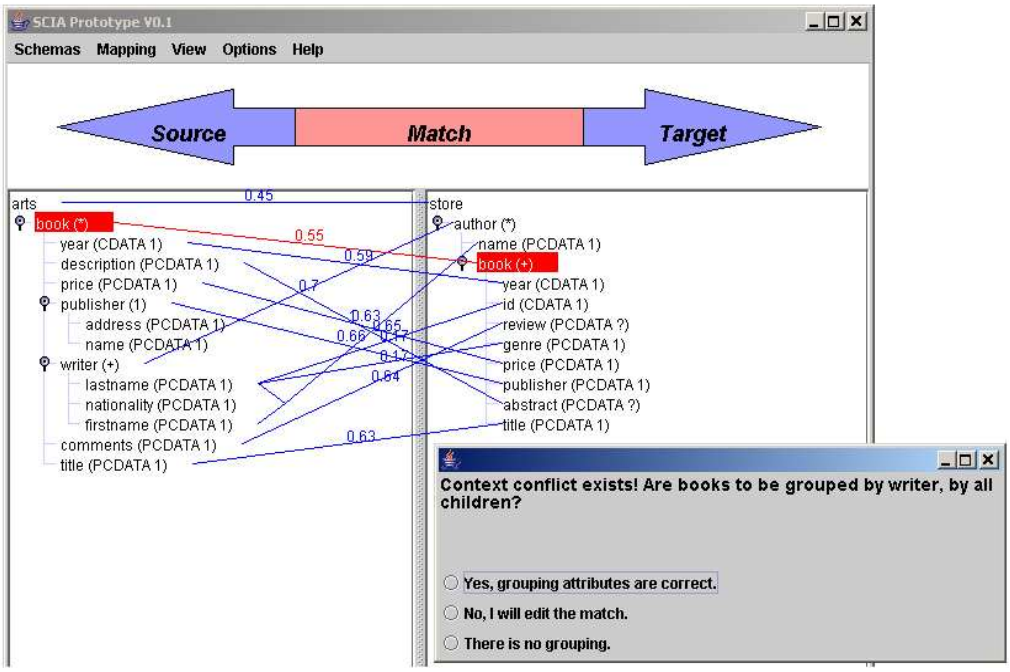

Figure III.4 A crtical point for potential grouping books by author

wrong, for example, publisher's `name` matched to author's `name`, writer's `lastname` to book's `id`. Some matches need more information, for example, from `arts/book/book` to `store/author/book`, a condition for correct grouping is needed. Figure III.3 and III.4 show two sceenshots of crtical points during context check. Figure III.5 shows the final matching results after context check. The genrated view is shown in the View Editor in Figure III.6, while the execution result, i.e., the translated document, is displayed in the Query Result Viewer in Figure III.7. Each step will be discussed in the following sections in detail.

III.C.1 Import Module

The input consists of target and source schemas, and optional domain thesauri. Schemas include DTDs, XML Schemas, relational schemas, and also ontologies. Schemas can also be extracted from certain standard metadata documents that provide schema information for the underlying data, for example, EML (Ecological Metadata Language) and ADN (ADEPTS/DLESE/NASA's joined digital

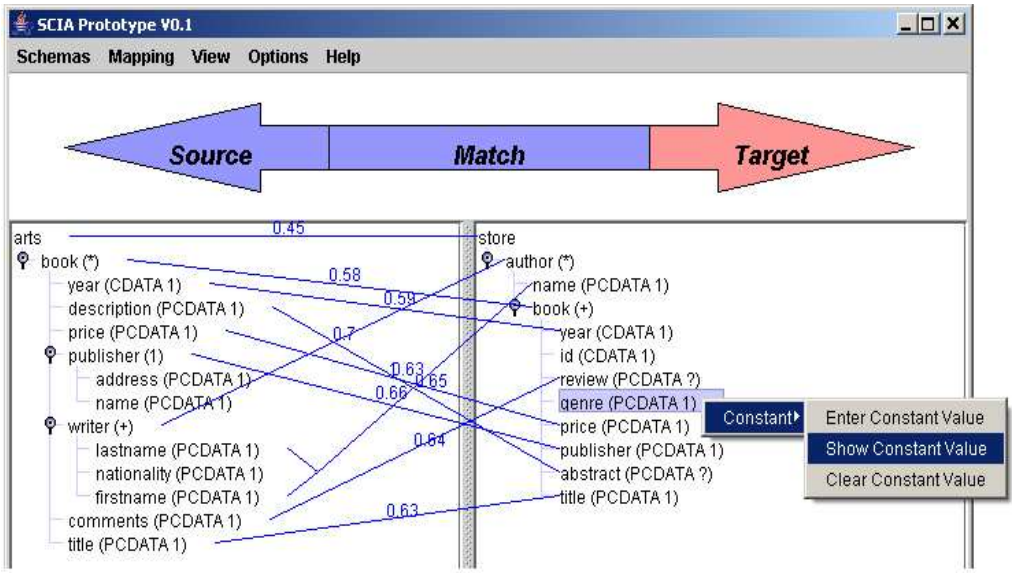

Figure III.5 Mapping results after context check

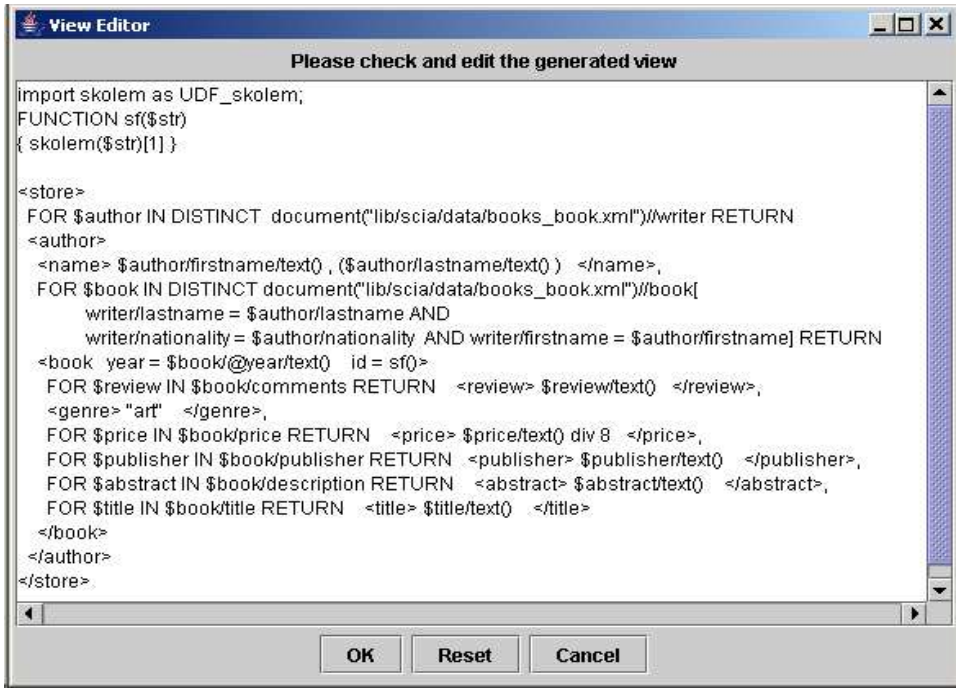

Figure III.6 Generated view in Quilt format

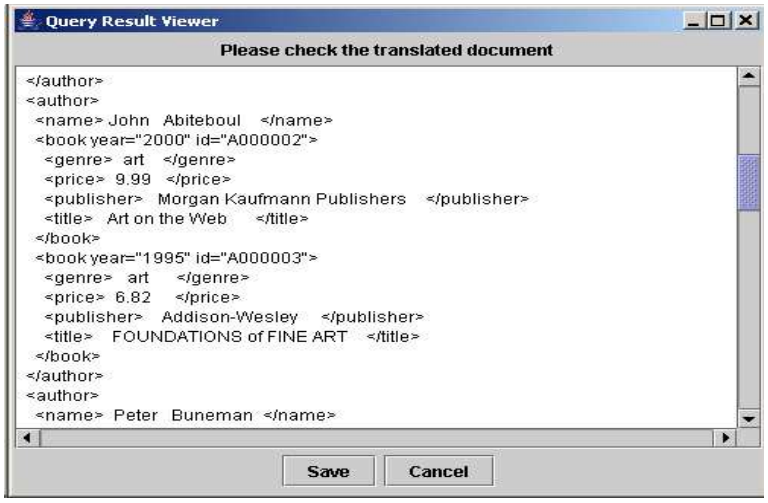

Figure III.7 Translated instance

library for geoscience). The **Schema Import** module produces a tree and a graph representation of each schema. In the tree representation, each element and attribute of a schema is a node, and sub-elements are child nodes. Each node has its type (attribute, leaf, or parent that is possibly a context), data type, height, constraints, two sets of matches (n_j, *sim*) for node n_j in the other tree, one for pre-context-check and another for post-context-check, etc.

The graph representation uses the RDF (directed labeled graph) model. Each element and attribute that is a node n_i becomes an RDF resource, res_{ni}. Each node type nt_i and data type dt_i become RDF resources res_{nti} and res_{dti} respectively. Corresponding edges are created for each node and added to the graph, two for its parent if it is not the root node, (res_{ni}, $parent\_height_{ni}$, $parent_{ni}$) and ($parent_{ni}$, $child\_height_{ni}$, res_{ni}), one for its node type, (res_{ni}, *nodeType*, res_{ntni}), one for its data type, (res_{ni}, *dataType*, res_{dtni}), etc.

III.C.2 Name, Path and Data Type matching

In the current prototype, linguistic matching over name and path includes: (1) terminological matching, in which element name and path are normalized through domain specific thesauri containing a synonym table, an abbreviation

table and a hyponym table; and (2) syntactic matching, which is purely based on string similarity.

Regarding syntactical matching, we first used the implementation of string matcher based on longest common substring in [63]. Now we use a letter pair string matcher, which finds out how many adjacent character pairs are contained in both strings, to solve problems with existing algorithms, such as Soundex Algorithm, Edit Distance, and Longest Common Substring. By considering adjacent characters, not only the characters, but also the character ordering in the original string are taken account, since each character pair contains a little information about the original ordering, which takes account both the frequency and ordering of letters [98].

For any pair of nodes (n_i, n_j) in the two trees, both the normalized name similarity value $ns_{i,j}$ and the normalized path similarity value $ps_{i,j}$ are computed,

$$ns_{i,j} = Coeff \times StringMatch(Normalize(name_{ni}, name_{nj}))$$
$$ps_{i,j} = Coeff \times StringMatch(Normalize(path_{ni}, path_{nj}))$$

where $Coeff = 0.8$ if $name_{ni}$ or $name_{nj}$ happens to be hyponymic to the other, otherwise $Coeff = 1.0$.

$$dts_{i,j} = Compatibility(dt_{ni},\ dt_{nj})$$

The name, path and data type similarity value, $ls_{i,j}$, is the weighted sum of name, path and data type similarities,

$$ls_{i,j} = ns_{i,j} \times w_{n\_} + ps_{i,j} \times w_{p\_} + dts_{i,j} \times w_{dt\_}$$

where $w_{n\_}$,$w_{p\_}$ and $w_{dt\_}$ are the weight values for name, path and data type similarity respectively, at the linguistic and data type matching steps, and $w_{n\_} + w_{p\_} + w_{dt\_} = 1.0$.

If there are multiple source nodes with big subtrees having almost the same name, path and data type similarity to a target node that also has a big subtree TST, and if these subtrees are also similar, based on their share of linguistically similar nodes, then this might indicate that the target subtree TST potentially corresponds to multiple source subtrees. Questions about whether there

exist multiple match subtrees for this target subtree, which of the similar ones are good ones, and whether there are more, should be posed to users. After user input on matching subtrees, SST_i ($1 \leq \mathrm{i} \leq \mathrm{k}$), for target subtree TST, the original problem of matching source schema tree S and target schema tree T is divided into k+1 smaller matching problems: matching $T - TST$ with $S - \sum_i SST_i$, matching TST with each SST_i for $1 \leq i \leq k$.

III.C.3 Description Matching

Sometimes, the names of schema elements are not descriptive, especially those of workflow messages, for example, elements are often called in1, in2, out1, out2, etc., although they may have associated descriptions. See Figure IV.9 for examples. So in our linguistic matching module, we also provide a description matcher to compare two schema elements by computing their textual description similarity. Our description matcher uses the Apache Lucene information retrieval library [27], which is based on Vector Space Model [44], and refers to the strategy in [84].

First, the textual description is normalized by removing stop words, affixes, prefix, and applying stemming techniques. Second, extract the terms' vector containing every term with its associated term frequency in the description. Then, compute the cosine of the two terms vectors to evaluate a part of the pair wise description similarity. This takes into account only the term frequency in both descriptions, experiments show it is not sufficient to determine the description similarity. Hence we add another computing to this description comparison. Supposing that each description associated with every element of the target schema forms a document, the set of these documents create a corpus which is indexed. A query, i.e, the description extracted from a source element is normalized first, then compared with the descriptions in the corpus by searching the above index. A set of scores are returned, which indicate how much the query is relevant to the descriptions in the corpus. The query type we handle takes into account the terms

order in the description. Finally the score and the vector cosine are combined as the description similarity between two given elements.

$$ds_{i,j} = DescriptionMatch(Normalize(description_{ni}, description_{nj}))$$

III.C.4 Structural Matching

The Similarity Flooding algorithm for graph matching in [62] was extended to structural matching for our prototype. Taking as input graphs G_t, G_s, and a set of initial similarity values, *InitMap*, between the nodes of the graphs, it iteratively propagates initial similarity values of nodes to surrounding nodes, using the intuition that neighbors of similar nodes are similar, and finally returns the structural similarity $gs_{i,j}$ of any node n_i in G_t and any node n_j in G_s.

$$gs_{i,j} = GraphMatch(G_t, G_s, InitMap)$$

The original algorithm was modified to freeze input mappings from the user and/or of high similarity from previous steps for some nodes. The initial mapping *InitMap* includes all user input matches, and the best candidate with $max\_ls_i \geq th_{l\_g}$ for each target node from the linguistic and data type matching step, where max_ls_i is the highest similarity value from the linguistic matching step for the target node n_i, and th_{l_g} is the linguistic similarity threshold at the graphical matching step. The quality of *InitMap* has a very important influence on the output of the structural matcher; users can also improve quality by confirming good matches, denying bad ones, and adding new ones. The following shows how this works. The s_bib.dtd and s_arts.dtd in Figure III.8 are simplified to provide more readable graphs. Their representations are shown in Figure III.9 (the edge label *cl1* stands for "child at level 1", *dt* for "data type"). Here the *InitMap* contains `([bib/book/title, arts/book/title], 0.77)` from the linguistic matching step, and built-in matches for DTD node and data types, such as ([`parentElem, parentElem`], 1.0), ([`leafElem, leafElem`], 1.0), ([`attribute, attribute`], 1.0), ([`PCDATA, PCDATA`], 1.0), ([`CDATA, CDATA`], 1.0).

Figure III.10 (a) shows the pairwise connectivity graph. The positive

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title)>
<!ATTLIST book year CDATA #REQUIRED >
<!ELEMENT title CDATA>
```

```
<!ELEMENT arts (book| article) >
<!ELEMENT book (title)>
<!ATTLIST book date CDATA #REQUIRED >
<!ELEMENT title CDATA>
<!ELEMENT article (title)>
<!ATTLIST article date CDATA #REQUIRED >
```

Figure III.8 s_bib.dtd and s_arts.dtd

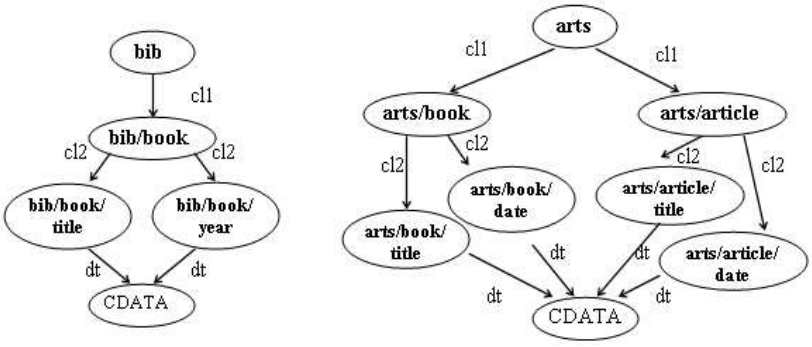

Figure III.9 The graphs for s_bib.dtd and s_arts.dtd

value on top of a matching pair node is its initial similarity, and is zero if not shown. The propagation graph in Figure III.10 (b) is induced from this pairwise connectivity graph (but the added propagation edges of the 7 nodes at lower part are not shown, in order to reduce the complexity). For each edge (*source, label, target*), an opposite edge is added for propagating *target* node's similarity to the *source* node. The edge *label* is changed to a *weight* value that indicates how much of the similarity value of the source node will propagate to the target node in each iteration. The *weight* value is determined by the number of outgoing same *label* edges from the *source* node, and whether the different target nodes are treated equally or weighted. Here we treat all targets equally, so the weight for each outgoing same label edge is 1.0 divided by the number of these edges. Several fixpoint formulas were explored in [62]; in the basic one, after each iteration, the map pair node similarity $sim^{n+1} = Normalize(sim^n + \sum_i weight \times sim_i^n)$ for each incoming edge e_i, where sim_i^n is the similarity value of the incoming edge e_i's source node at step n. The normalization is done by dividing by the maximal similarity value among all map pair nodes at the current step. Propagation continues until

the Euclidean length of the residual vector $\Delta(sim^{n+1}, sim^n)$ is less than threshold. We made the following modifications: (1) "good" map pair nodes propagate their similarity to neighbors, but neighbors don't propagate to them, which means that the propagation formulae are applied only to "non-good" map pair nodes; and (2) "good" matching pair nodes don't join the normalization process.

In Figure III.10 (b), the final similarity values (*sim1, sim2*) are shown around each map pair node, where *sim1* is from the original algorithm without separating shared elements in the input graph, and *sim2* is from the modified algorithm that fixes high similarity initial mappings and uses paths instead of elements to make shared elements context dependent. The second gives higher similarity values for correct matches and lower similarity values for most wrong matches, although it doesn't differentiate `([bib/book, arts/book], 0.88)` from `([bib/book, arts/article], 0.88)`, the later combination of linguistic and structural similarity makes `(bib/book, arts/book)` win significantly. The first method doesn't differentiate `([year, title], 0.42)` from `([year, date], 0.42)`, since they have the same similarity value, and it gives too low similarity to `([bib, arts], 0.29)`; both cases cause difficulty on filtering.

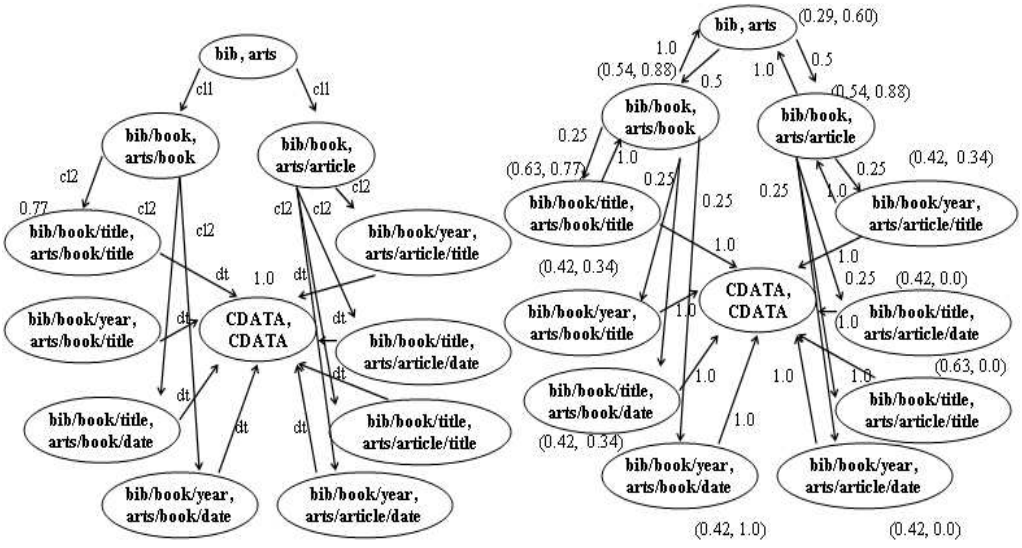

Figure III.10 (a) Pairwise connectivity graph and (b) Induced propagation graph

After this graph matching step, the similarity between two tree nodes is the weighted sum of similarity values of name, path, data type, description and structure:

$$s_{i,j} = ns_{i,j} \times w_{n\_c} + ps_{i,j} \times w_{p\_c} + dts_{i,j} \times w_{dt\_c} + ds_{i,j} \times w_{d\_c} + gs_{i,j} \times w_{g\_c} \quad (*)$$

where w_{n_c}, w_{p_c}, w_{dt_c}, w_{d_c}, and $w_{g_c}$are the weight values for combining the name, path, data type, description and structural similarities to identify the best match candidates for each node right before the following context check step, and $w_{n\_c} + w_{p\_c} + w_{dt\_c} + w_{d\_c} + w_{g\_c} = 1.0$.

III.C.5 Context Check

The basic matches predicted automatically are refined by checking and re-matching subtrees rooted at core elements. Since tag meanings vary with context, and context is given by higher elements, we seek to identify **core** contexts and attributes, the most important contextualizing elements for tags within subtrees, by using heuristics and user input; then threshold and context are checked for them. For example, in matching two book DTDs, the contexts `/arts/book` and `/arts/article` are found to be core contexts, with title, author, publisher as their main attributes. Previous steps found that the best match with `/arts/article` is `/bookstore/book`, but its similarity value is lower than the core threshold, so that matches in its subtree are not reliable, and user input is needed. Figure III.11 shows the screen shot at this critical point, while Figure III.12 shows the result after the user selects the "No match" option for that critical point, with all wrong matches for the article subtree cleaned up automatically.

After basic matching is computed, core contexts are identified. Then the core contexts are sorted in ascending order of height from the root, for top-down matching refinement. For each node n_i, check if its best matching node has maximum similarity bigger than a relatively high similarity threshold. If not, for this important target node, there may be no good match from the combined matching steps, so that its subtree matches also might not be good, and user interaction is

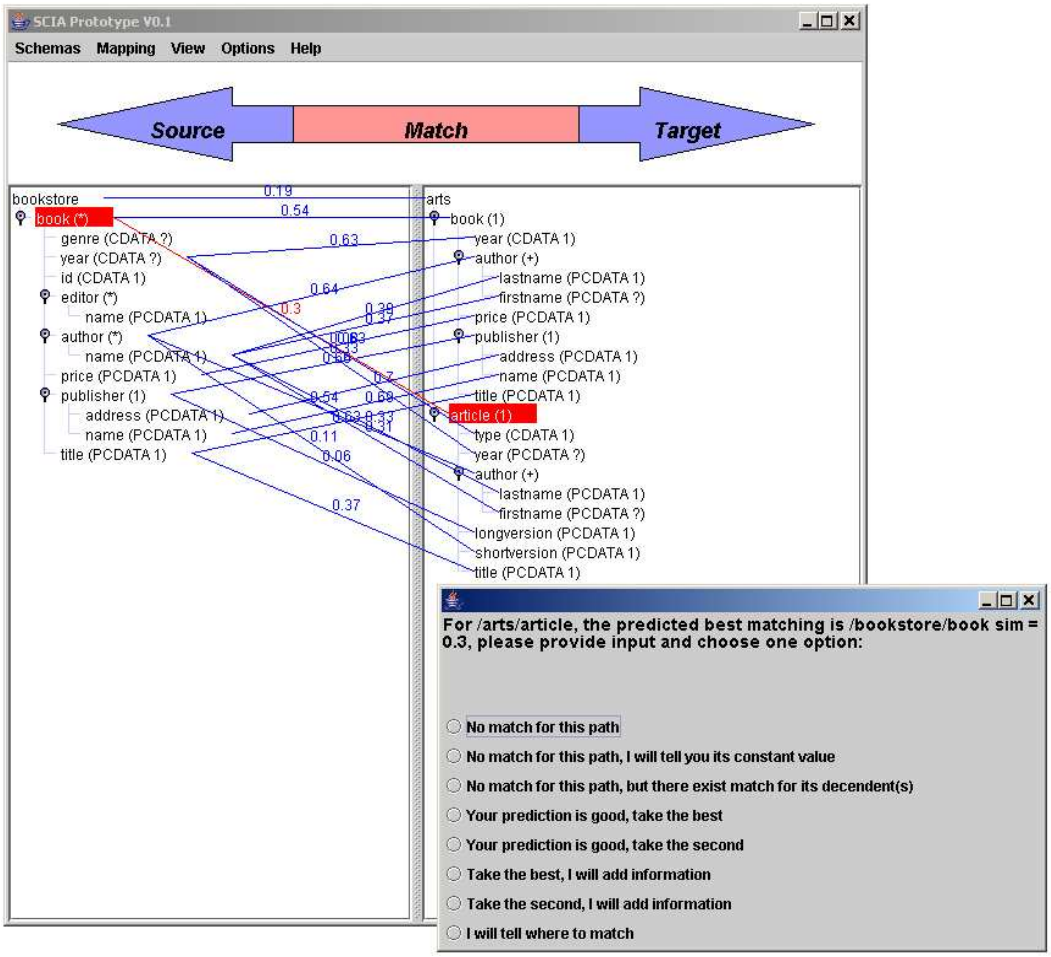

Figure III.11 Interface at a critical point for `article`

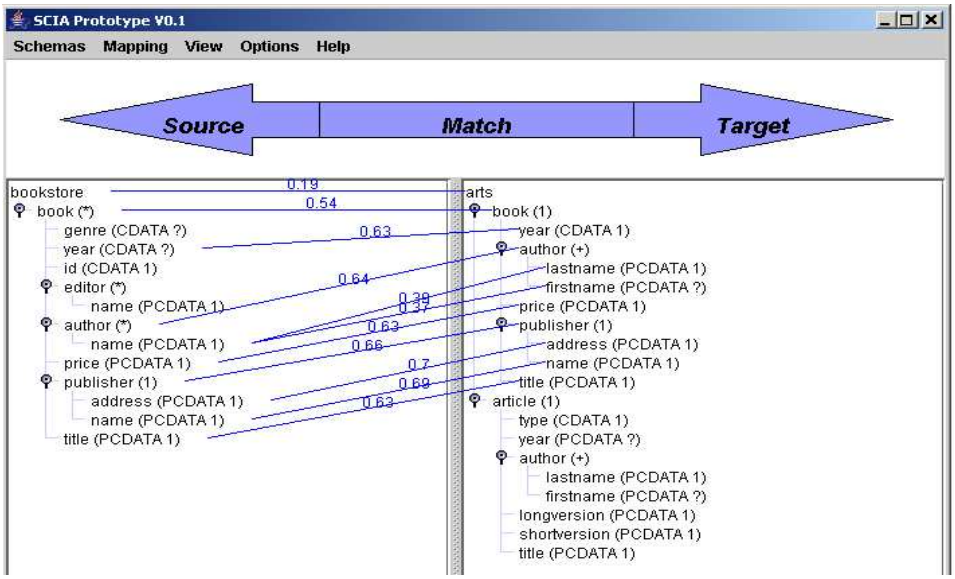

Figure III.12 Mapping results after resolving the critical point `article`

requested; or in automatic mode, some matches with very low similarity within the subtree are thrown away, and child nodes that are core nodes are checked further. If a good match is found, refinement of subtree matches to best matching node n_j's subtree starts by computing initial mappings, $InitMap_{subtree\_ni,subtree\_nj}$ which are the best so far within these two subtrees (but not necessarily best at whole trees) and with similarity bigger than a threshold. Then subtree graphs $G_{subtree\_ni}$ and $G_{subtree\_nj}$ are generated as input with $InitMap_{subtree\_ni,subtree\_nj}$ together for *GraphMatch* to compute again the structural similarities, $gs_{h,k}$ of nodes n_h in n_i's subtree and nodes n_k in n_j's subtree, by

$$gs_{h,k} = GraphMatch(G_{subtree\_ni}, G_{subtree\_nj}, InitMap_{subtree\_ni,subtree\_nj})$$

Not only the children of core contexts are recomputed according to the user feedback, but also the ancestors. In current implementation, we only propagate the change of a core context to its ancestors when it gets a better match, see the change of similarity values between `arts` and `store` from Figure III.4 at a critical point for potentially grouping `book` by `author` to Figure III.5 after recomputing by taking user confirmation of the correspondence and corresponding grouping attributes at

this critical point and others. More checking is needed to see whether any almost equally good multiple matches with very high similarity values from the linguistic and data type matching step for these core nodes are still competing candidates after context check. If they are very different, this might indicate that some multiple matches would be missed; user input here can avoid missing multiple good matches. Then the core node subtree is matched to the multiple matching subtrees as discussed above, and a new similarity value for n_h and n_k is computed using the formula (*), but the weight values for similarities from name, path, data type and graph matchers may change. Then for each target node n_i, another set of matches (n_j, sim), where n_j is a node in the source tree, is computed as the matching result of the context check matcher.

Core nodes might not group at only one or two levels in a schema tree, and they might not be identified precisely by heuristics. User input on specifying some or all core nodes can help the system concentrate on checking important subtree matches. When it is hard for the system to decide good matches for a core node, user feedback can significantly improve whole subtree matches, and significantly reduce total user effort.

III.C.6 Combination of Mapping Results

After performing the above matching steps, each target node gets a set of candidates from combining the results from the four individual name, path, data type and structural matchers, and another set of candidates from the context check matcher. Selecting top matches from one set, or from combination of both sets, depends on how they differ from each other. If the context check results win significantly with comparison based on the best match similarity value, the context check results will be selected, otherwise the pre-context-check results are selected. If they are similar, user input is requested, or by default the top candidates from both sets are selected by sorting them together non-interactively.

Our experiments and prior work [17] show that the direction of matching

also affects the quality of matching result. Users can choose selecting match results from a single direction or both directions based on their knowledge of the schemas. Both-direction means matching target to source and also matching source to target, if a correspondence (n_t, n_s) in the target to source mapping is selected only if target node n_t is one of top k (e.g., k = 2) almost equal candidates for source node n_s in the source to target mapping. Both-direction is the default strategy, since it performs better on the average.

User input can also help decide some important parameters, such as 1) how many best match candidates should be selected (by default only the best one is selected); 2) the threshold for selecting or throwing matches with lower similarity values for each target node, and 3) what delta value for deciding multiple candidates with almost equal similarity values in order to detect real multiple matches for output.

Through the graphical user interface, the user can also specify necessary operations and conditions if they can not be automatically recognized or have not been specified earlier. Since without formal metadata, ontological information or content information, it is nearly always infeasible to recognize precise conversion functions, developing a high quality user interface to support directly specifying them is more helpful than asking users to provide formal metadata or domain ontologies as hints to discover them.

III.C.7 View Generation

The SCIA mapping contains information on how each target node is populated from source side data, or generated by function (including constant), and necessary conditions and grouping attributes for each target node. Executable Mapping Generator walks the target schema tree, generates join and grouping conditions for whole tree based on structure, constraints, data type, cardinality, and mapping, especially the conditions, grouping attributes and functionsobtained through interactions with the user; generates executable view definition describing

the target schema in terms of the source schema; asks for user input if information is not adequate.

Depending on the application, different formats of view defitions can be generated, e.g., XSLT for pure data transformation, or XQuery for both transformation and integration; XQuery is used in our current prototype. An algorithm has been developed and tested, which takes as input a target schema tree with mapping information to a source schema tree, and outputs a query that can be executed to transform a source document conforming to the source schema to a document conforming to the target schema. It is briefly described by the following pseudo code:

```
generateView():
  if this schema tree node n is a leaf node
    case 1: n has independent local matches
      for each local match
        view = view+getViewFromLocalMatchForLeaf();
    case 2: n has dependent global matches
      for each local match: match
        if each node in match is descendent of
            cAncestorMatch node(s)
          view = getViewFromLocalMatchForLeaf();
    case 3: n has only local match
      view = getViewFromLocalMatchForLeaf();
    return view;
  set view tail = END_TAG; e.g., "</Seq-data>"
  if n is root
    head = START_TAG; e.g., "<Sequences>"
    middle = getChildrenView();
  if n is non-root parent
    head = FOR_CLAUSE + "RETURN";
    (e.g., "FOR $segment in $sequence/segment
         RETURN")
    case 1: n has independent local matches
      for each local match: match
```

```
        middle = middle + getChildrenView();
        if match is the last
          middle = middle + tail;
        headAndMiddle = headAndMiddle + head +
attrStr + middle;
reset middle and attrStr;
    case 2: n has only local match:match
      middle = middle + getChildrenView();
      headAndMiddle = head + attrStr + middle;
  view = headAndMiddle + tail;
  return view;

getChildrenView():
  for each child
    view = view + generateView();
  return view;

getViewFromLocalMatchForLeaf():
  if this leaf node n is an attribute node
    only update parent attribute part in view;
  if n is a leaf element without any attribute
    case 1: local match, 1:1 without conditions
      head = FOR_CLAUSE + "RETURN";
      extract the function expression in match;
      view = head + START_TAG + current bound
             variable + "/text()" + function
             expression + END_TAG;
    case 2: multiple source nodes, no conditions
      deal with multiple match units in order and
           update view;
    case 3: involves conditions
      get appropriate ancestor variable by
           checking nodes involved in conditions;
      add more FOR_CLAUSE into view;
      organize WHERE_CLAUSE and add into view;
```

```
return view;
```

III.C.8 Handling Large Complex Schemas

More challenges arise for handling complicated schemas, like the EML (Ecological Metadata Language) schema, and ontologies in SCIA. First, recursive references cause infinite loops for building schema trees. Secondly, the huge structural differences between schemas break the common heuristics used in structural matchers, for example, the intuition that similar nodes' neighbors are also similar does not help with matching EML with ADN, the metadata standard for earth system sciences. Moreover, performance becomes a serious issue.

We address the first and the third problem by providing a mechanism that allows users to expand, remove, and hide from matching recursive nodes and regular schema subtrees. For example, in EML schema, the dataSource is type of DatasetType, the same as the type of its ancestor DataSet, which is a recursive reference. By default, the system does not expand it, instead users can expand it as many levels as they want in the GUI. In addition, usually only a small portion of a complex schema or ontology would be used in practice for a specific application, for example, an EML file typically contains only a dataset and datatable, while there are about 20 more modules, such as access, constraint, coverage, literature, methods, project, protocoal, resource, software. It would not help with anything to match the whole EML schema tree to another schema. Through the GUI, users can trim schema trees easily, to make the mapping task more manageable.

We address the second issue by using a more flexible combination strategy for multiple matchers, e.g., adjusting the weights for combining the similarity values for results from different matchers according to the mapping task properties. More specifically, when structural difference between schemas is huge, the system can give higher weight values to name matcher and context matcher, but lower weights for path matcher and structural matcher.

III.D Biodiversity Application

Biodiversity, the richness of life, results from complex interactions among many species, abiotic ecosystem factors, generic factors, and multiple ecosystems [50, 87]. These interactions occur at very different scales: molecular and cellular (e.g., genetic makeup), organism (e.g., specimen), and ecological (e.g., spatial distribution of species and global climate events). Hence biodiversity studies involve complex data from molecular biology, natural history collections, field ecology, remote sensing, etc. There are also communication and coordination issues involving different groups, interests, regions, backgrounds, points of view, etc. Data accuracy, precision, resolution, and scale can vary greatly, and different conventions for representing and naming data are common. Biological and ecological data are often geo- and species-referenced in incompatible ways that change over time. Many key biodiversity questions involve change in range, number, distribution, genetics, and proportion, e.g, extinction, migration, incursion, restoration, and environmental impacts. Historical information is needed to analyze trends, adaptations, and long-term relationships. All this greatly complicates data integration and analysis.

The following are typical modern ecological queries involving both macro- and micro-level relationships among various species and between various biodiversity and ecosystem factors; processing of Query 1 is discussed in Section III.D.1.

Query 1: Find DNA sequence data for all species in site North Temperate Lakes with density greater than 13.0 per km^2.

Query 2: Find the DNA sequences of species sharing a common habitat with Picea rubens and having density difference less than 0.5 per km^2 in that habitat.

Query 3: Find the density values of species Picea rubens and total number of all species in certain sites from year 1973 to 2003 and monthly average temperature and precipitation of these sites from 1972 to 2003.

III.D.1 Sample Schemas

We tested our tool by integrating some data sources relevant to such queries. The sources for genetic sequence information are in the formats of BSML (*bs*) [49], GAME (*g*) [54] and BIOML (*bi*) [57]; the sources for species distribution are from two different ecological studies, studyA (*sa*) and studyB (*sb*) [26]. A target sequence DTD (*s*, `sequence.dtd`) was created for genetic sequences and a XML Schema (*o*, `obs.xsd`) was designed as the target for ecological observation according to discussions in [26]. Only these two target schemas are exposed to end users for querying the integrated database, helping them to organize valid queries to get the information they want without needing to know details of the five underlying data sources; for example, Query 1 might look as follows:

```
FOR $Seq in document(sequence.xml)//Sequence
 FOR $Obs in document(obs.xml)/Observation
 WHERE $Seq/Organism = $Obs/Species AND
    $Obs/Site = "NTL" AND $Obs/Density > 13.0
 RETURN <Species>
   <name> $Obs/Species/text() </name>
   <Seq-data> $Seq/Seq-data/text() </Seq-data>
   </Species>
```

The Sequence DTD and observation and studyA XML Schemas are below, beginning with `sequence.dtd`:

```
<!ELEMENT Sequences (Sequence*)>
<!ELEMENT Sequence (Description?, Seq-data?,
   Segment*, Reference*, Organism?)>
<!ATTLIST Sequence
   name CDATA #IMPLIED
   id CDATA #IMPLIED
   length CDATA #IMPLIED
   type (mol-not-set | dna | rna | other-mol)
    "dna" genomeref IDREF #IMPLIED ... >
<!ELEMENT Organism (#PCDATA)>
```

```
 <!ELEMENT Seq-data (#PCDATA)>
 <!ELEMENT Segment EMPTY>
 <!ATTLIST Segment
    seg-source CDATA #IMPLIED
    seg-id CDATA #IMPLIED
    seg-role CDATA #IMPLIED
    seg-start CDATA #IMPLIED
    seg-end CDATA #IMPLIED ... >
 <!ELEMENT Reference (RefAuthors?, RefTitle?,
      RefJournal?)>
...
```

StudyA and Observation have the structures below:

```
  StudyA                    Observations
   Obs                       Observation
     Date      string          Study    string
     Location  string          Date     string
     Species   string          Site     string
     Count     integer         Species  string
     Area      float           Density  float
```

Some ecologists in the SEEK project desire another format for the integrated observation data, which is shown below, along with another source schema, extracted from the LTER GCE site EML file.

```
  GCE site schema                LTER global schema


  Table                          LTER-Observation
    Row                            Tuple
      Site          string           Station       string
      Community     string           Year          integer
      Treatment     string           Season        string
      Replicate     integer          Locality      string
      Species_Code  string           Block         string
      ITIS_TSN      string           Treatment     string
      Plant_Mass    float            Plot          integer
```

```
Plant_Mass_m2 float              Biodiversity  float
                                 Productivity  float
```

The main challenges in LTER data integration problem include:

1. Different LTER stations use different naming conventions. E.g., "site" in one schema may refer to a LTER station, like GCE or ARC, or a study site within a station, like GCE6 as a block.

2. At different LTER stations, measurements are done at different levels, at the plot (square meter) or sub-plot level. For example, at GCE, measurements are done at quadrant (0.25 square meter) level, and multiple rows need to be grouped into a single tuple in the integrated data set.

3. Indirect matches are involved because biodiversity observation at different granularity are recorded. For example, GCE records each species code, where that species was found and other information as a row, but in the LTER integrated data set, only the number of distinct species that occur in a plot is needed. Another example is that StudA has separate attributes for species count and the area of the location in which the species occur, but Observations has density, which equals to count divided by area. In addition, some required elements don't have corresponding elements in the source schema, but their value can be specified as a constant, or obtained from the EML file by the user.

Therefore, aggregation and conversion functions are required in the schema mappings due to 2) and 3). However, the information for correct aggregation and conversion are hard to specify formally by data provider, and informal specification is difficult for machine use.

III.D.2 Results

We ran these tasks in different modes and compared matching accuracy values, these are discussed in Section III.F. From the match results for each pair of

a target and a source schema, an individual view of each target over each relevant source is generated, and then an integrated view of each target over all relevant sources is obtained as a union of the individual target views. Some fragments of individual views generated from the match results follow; the first is the view of the sequence schema over the BSML schema:

```
<Sequences>
 FOR $Sequence IN  document("bsml.xml")//Sequence
 RETURN <Sequence
  name = $Sequence/@locus/text()
  type = $Sequence/@molecule/text() ... >
 FOR $Seq_data IN $Sequence/Seq-data
 RETURN <Seq-data> $Seq_data/text() </Seq-data>,
 FOR $Segment IN $Sequence/Segment RETURN
  <Segment
   seg-start = $Segment/@seg-start/text() ... >
  </Segment>,
  FOR $Segment IN $Sequence/Feature-tables/Feature-
     table/Feature
  RETURN <Segment seg-start = $Segment/Location/
    Interval-loc/@startpos/text() ... >  </Segment>,
  ...
  FOR $Genome IN  document("bsml.xml")//Genome
   FOR $Organism IN $Genome/Organism
   WHERE $Genome/@id = $Sequence/@genomeref
   RETURN <Organism> $Organism/text()</Organism>
 </Sequence>
</Sequences>
```

The view of the observation schema over studyA:

```
<Observations>
 FOR $Observation IN  document("studyA.xml")//Obs
 RETURN
  <Observation>
  ...
```

```
   FOR $Site IN $Observation/Location
   RETURN <Site> $Site/text() </Site>,<Density>
      $Observation//Count/text() DIV
     ($Observation//Area/text()) </Density>
  </Observation>
 </Observations>
```

The view of the LTER global schema over GCE site schema:

```
<LTER-Observation>
    FOR $Site IN DISTINCT document("lib/scia/data/schema-PLT-GCED-
          0409.1.1.xml")//Site
    LET $Tuple:=  document("lib/scia/data/schema-PLT-GCED-
          0409.1.1.xml")//Row[Site = $Site]
    FOR $Treatment IN DISTINCT $Tuple/Treatment
    FOR $Replicate IN DISTINCT $Tuple/Replicate RETURN
    <Tuple>
        <Station> $Site/text()</Station>,
        <Year> "2000"</Year>,
        <Season> "FALL"</Season>,
        <Locality> $Site/text()</Locality>,
        <Block> $Site/text()</Block>,
        <Treatment> $Treatment/text()</Treatment>,
        <Plot> $Replicate/text()</Plot>,
        <Biodiversity>
          count($Tuple[Treatment = $Treatment and
                       Replicate = $Replicate]
          /Species_Code)
        </Biodiversity>,
        <Productivity>
          sum($Tuple[Treatment = $Treatment and
                     Replicate = $Replicate]
          /Plant_Mass * 4)
        </Productivity>
    </Tuple>
</LTER-Observation>
```

III.E Workflow Design Application

We have integrated SCIA into an open source system for scientific workflows, called Kepler, to help with workflow design and composition by suggesting connections and transformation steps between workflow components.

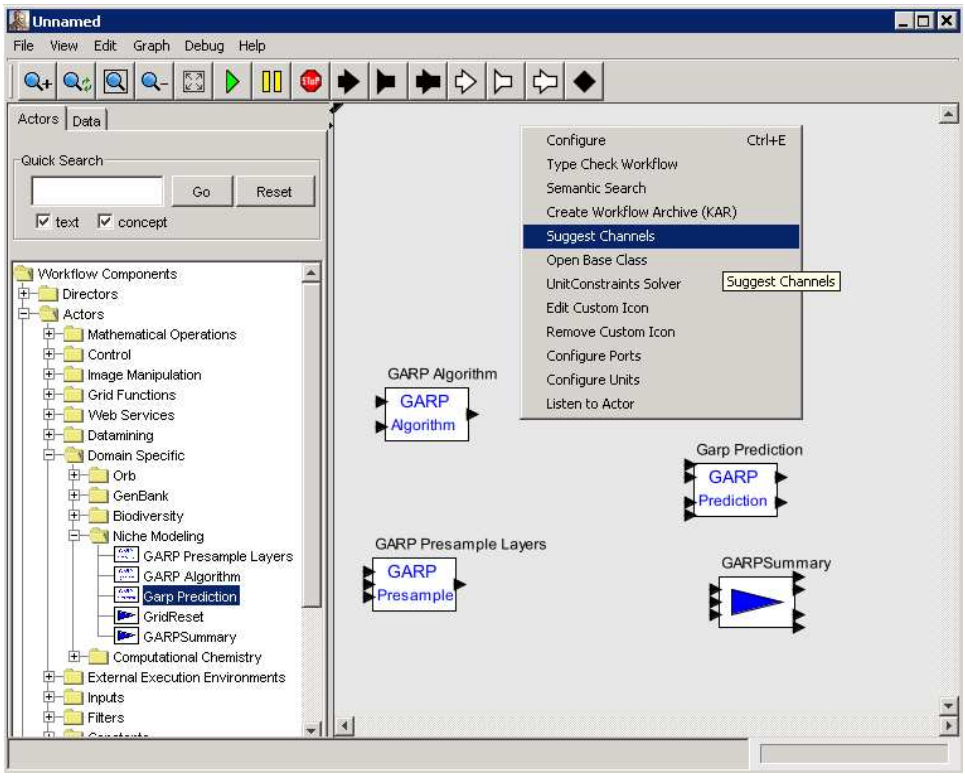

Figure III.13 The Kepler Graph Editor

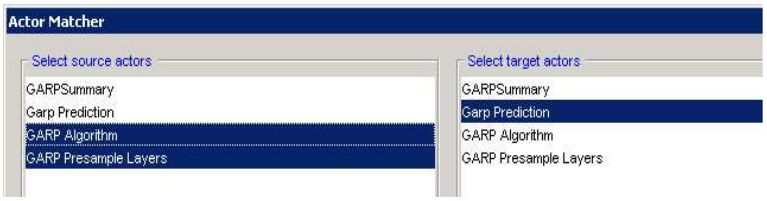

Figure III.14 Selecting source actors and target actors

Figure III.13 shows the Kepler workflow graph editor. Available workflow components, called actors in Kepler, are shown in the left side, which allow drag-drop into the main window on the right for designing workflows visually. Once the

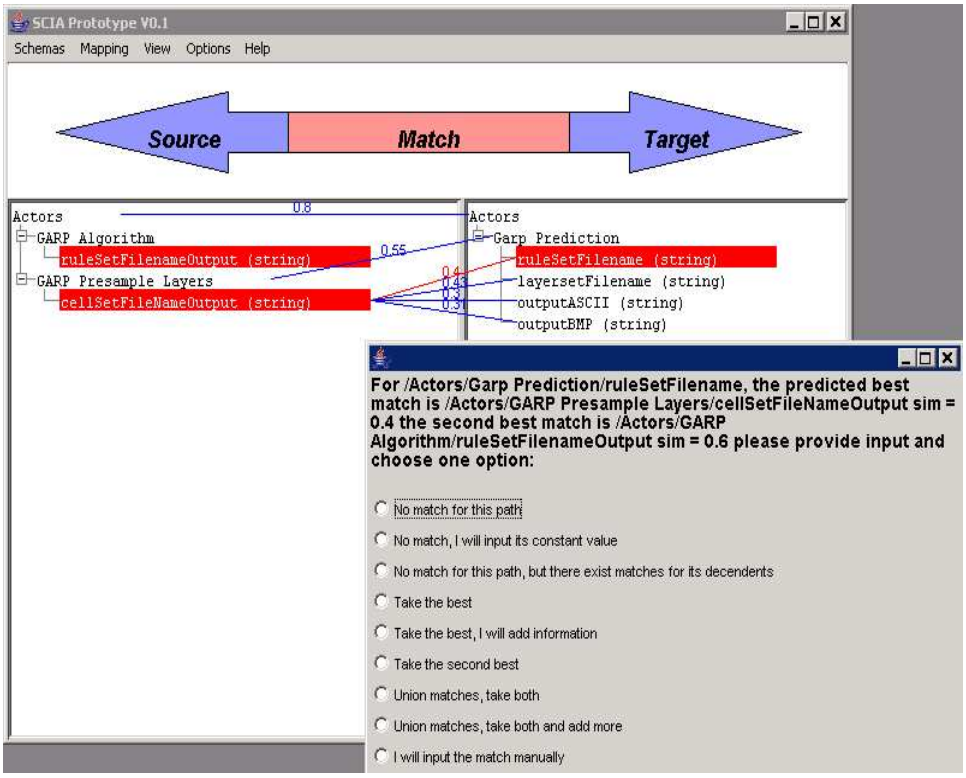

Figure III.15 A critical point for matching actors

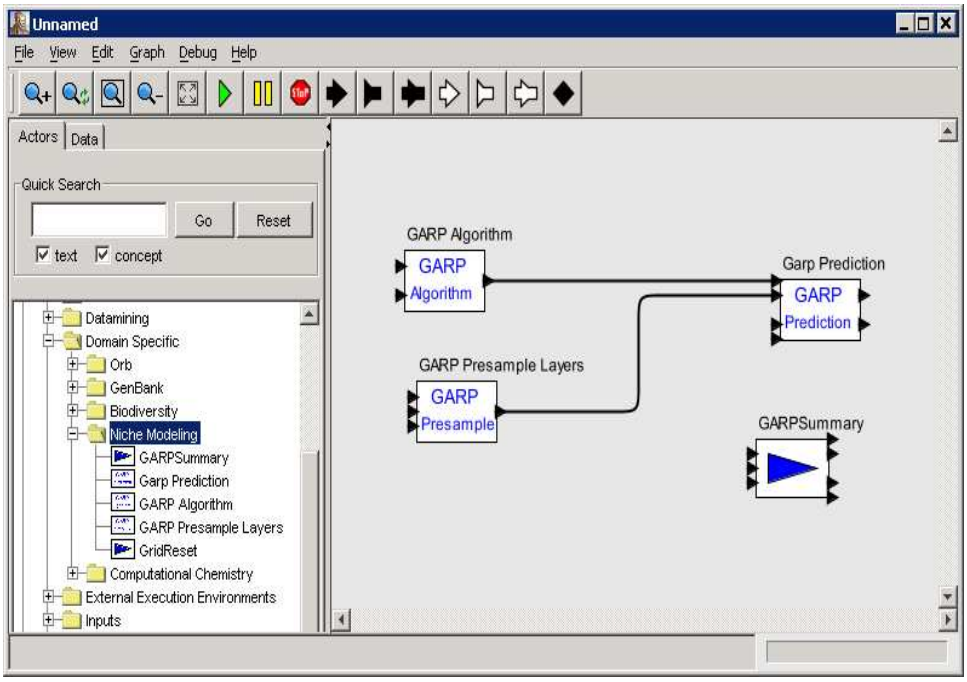

Figure III.16 The connected workflow after applying matches created through SCIA

user wants help with connecting components, The fifth entry "Suggest Channels" in the popup menu on the workflow graph window is for suggesting connections with components that appear in the workflow graph. After the source actors and target actors are selected, as shown in Figure III.14, they are imported as schemas in SCIA and can be matched as discussed for regular schemas. Figure III.15 shows a critical point for matching the selected source actors to the selected target actor for designing a workflow for niche modeling. After the correct matches are created, they can be applied to the workflow graph, resulting in the connected workflow in Figure III.16.

III.F Informal Evaluation

Experiments have been done for both qualitative and quantitative evaluations of SCIA. We start with traditional empirical metrics using real world schemas

from four application domains in this section and then formal evaluation in the following section.

III.F.1 Schemas Involved

Schemas from four application domains have been used to evaluate our approach: 1) three book DTDs obtained from the web, bibliography (*b*), arts (*a*), bookstore (*st)*, and a mediated one (*bk*), have been tested in our data integration prototype system DDXMI, using manual matching [66]; 2) three bio-molecular DTDs, GAME (*g*), BSML (*bs*), BIOML (*bi*), which have been trimmed to remove some branches that are huge but irrelevant to sequence encoding, or were caused by unsophisticated design, plus a mediated DTD (*s*) for gene sequence encoding only; 3) five XML Schemas for purchase orders, CIDX (*c*), Excel (*e*), Noris (*n*), Paragon (*p*), and Apertum (*ap*), used by COMA [17]; and 4) several DTDs/XML schemas or EML documents from LTER. The characteristics of relevant schemas are shown in Table III.1.

Table III.1 Characteristics of the tested schemas

Domain	*Bibliography*				*Biology*				*Business*					*Ecology*				
Schema	*b*	*st*	*a*	*bk*	*bs*	*g*	*bi*	*s*	*c*	*e*	*n*	*p*	*ap*	*sa*	*sb*	*o*	*gce*	*lter*
#Paths	13	7	21	13	69	124	32	30	40	54	65	80	145	7	8	7	10	11
#Nodes	11	7	15	13	58	66	21	30	40	35	46	74	80	7	8	7	10	11
Depth	4	4	5	5	10	7	8	4	4	4	4	6	5	3	3	3	3	3

III.F.2 Matching Quality Measures

A partial evaluation of the quality of our matching process can be obtained using the same measures used in [17, 62], derived from the information retrieval and data mining fields. However, as argued earlier, a metric that counts the total user effort would provide a much more appropriate evaluation, since our aim is to minimize this quantity, rather than to compete directly with systems that

are less interactive. The manually determined real matches R for a match task are compared with the matches P returned by the automatic matching process. The following are counted: the correctly identified matches T, the wrong matches $F = P - T$, and the missed matches $M = R - T$. These correspond to true positives, false positives, and false negatives respectively. Then the following three measures are computed:

$$Precision = T/P = T/(T + F)$$

$$Recall = T/R$$

$$Overall = 1\text{-}(F + M)/R = (T - F)/R = Recall \times (2 - 1/Precision)$$

The first gives the reliability of the mapping, the second gives the share of real matches found, and the third is a combined measure for mapping quality, taking account of the post-match effort needed for both removing wrong and adding missed matches.

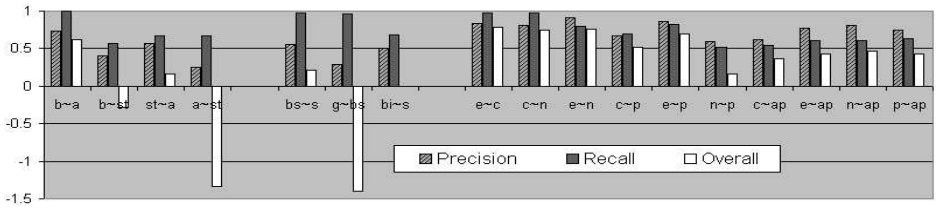

Figure III.17 Measure values: no context check, non-interactive mode, 1-direction

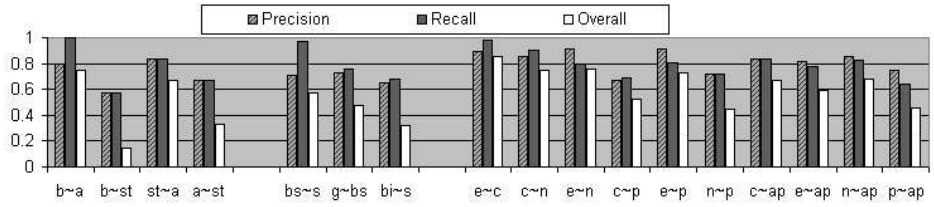

Figure III.18 Measure values: with context check, non-interactive mode, 1-direction

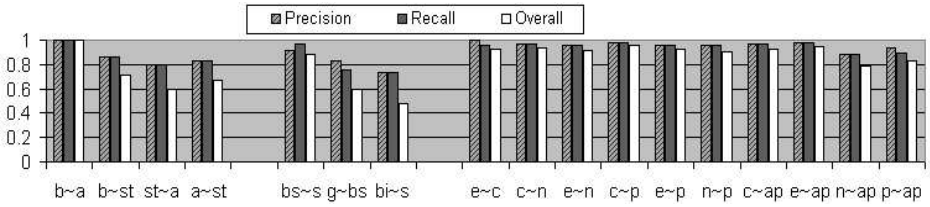

Figure III.19 Measure values: with context check, interactive mode, 1-direction

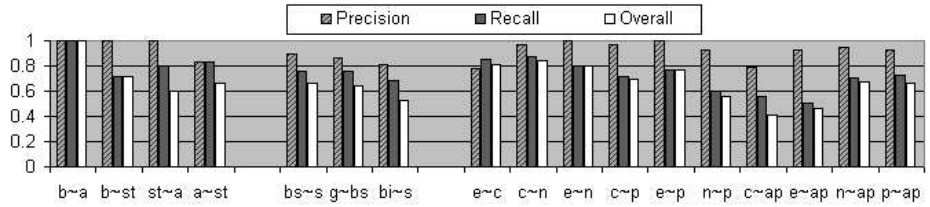

Figure III.20 Measure values: with context check, non-interactive mode, 2-direction

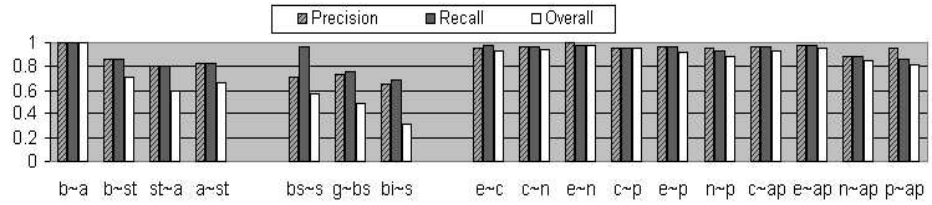

Figure III.21 Measure values: no context check, interactive mode, 2-direction

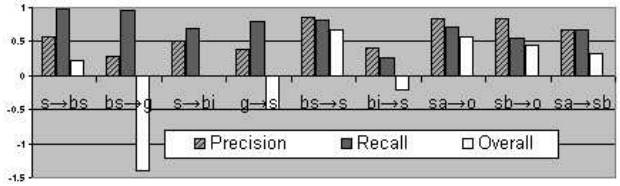

Figure III.22 Biodiversity application: no context check, non-interactive

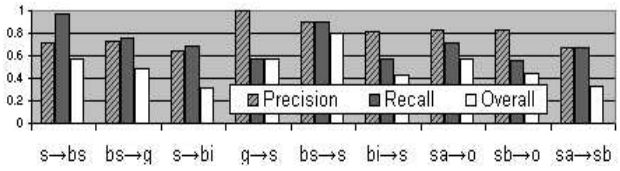

Figure III.23 Biodiversity application: with context check, non-interactive

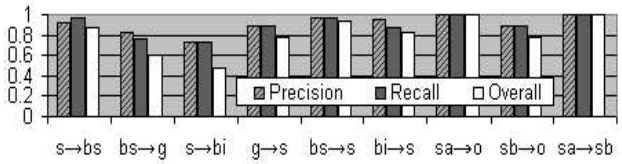

Figure III.24 Biodiversity application: with context check, interactive

III.F.3 Results

Figures III.17–III.24 show the values of these three measures for match tasks in these four domains, while III.22–III.24 are for some match tasks tested in different modes from the biodiversity application.

The preliminary results over the limited number of schemas show that context check and user interaction at critical points were very helpful for improving the accuracy of schema matches, andour approach works well compared with the purchase order schema tests used by Coma [17] and Cupid [59].

1. Using critical points improves matching accuracy. The overall values for the same tasks in Figure III.18 with context check for detecting and resolving critical points are significantly higher than those in Figure III.17 without context check. The results in these two graphs were obtained by running SCIA in the same non-interactive mode using 1-direction matching strategy. We also recognize that context check was especially helpful in 1-direction match, improving the *Overall* score significantly. Context check was very helpful for more complicated match tasks, e.g., significantly improved the *Overall* score of the tasks for complex bio-molecular schemas, by pruning large subtrees' lower-similarity matches, identifying multiple matches for core context elements. But it did not help the scores of four tasks for simple species distribution schemas, because all leaf elements in these schemas share the same context.

   User interactions at critical points resolved most complex matches, such as many-to-one or 1-to-0 matches, or those involving functions and/or conditions, e.g., correcting the mismatch of `Observations/Observation/Study`

to `StudyA/Obs/Location` and specifying its constant value for the name of the study where this data was obtained, pruning lower-similarity matches in multiple matched subtrees for matching BSML, GAME, BIOML to `Sequence`, editing local multiple matches and specifying their match expressions, for instance, `StudyA/Obs/Count` DIV `StudyA/Obs/Area` for `Observations/Observation/Study`. User interactions for context-holding elements were extremely efficient, saving much effort on matching their subtrees. For view generation, path contexts also helped to determine where to bind variables correctly for complex matches, especially when conditions and/or functions are involved. An informal observation is that user feedback at critical points during the mapping process with systematic help was much easier than editing the view afterwards.

2. SCIA outputs better matches than the most sophisticated schema matching tool COMA. Comparing the overall values of the purchase order schema matching tasks in Figure III.25 from COMA's best configuration (without reuse of existing schema matches) with Figure III.18 and Figure III.19, we conclude that in non-interactive mode, SCIA's matching performance is similar to COMA's best performance, while in interactive mode SCIA's matching results are much better than COMA's best results. Notice that some of those purchase order schemas use separate first name and last name for contacts, while others use full names. COMA regards a predicted match `name` → `first name` as correct. But SCIA regards it as a wrong prediction, because it does not say how to get `first name` from `name`. Therefore, based on the same real matches used by COMA, SCIA's measurement values for the purchase order schema matching tasks are better than those shown in the above figures.

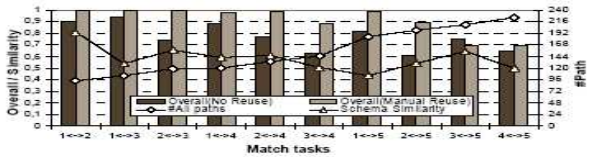

Figure III.25 COMA's best results

III.G Formal Evaluation

To quantify how much user effort it saves to use critical points in schema mapping, as well as to evaluate SCIA's user interface design, with Goguen, Zavelov and Rifaieh together designed, performed and analyzed the following formal experiments. In particular, Zavelov has been working with Goguen and me together on SCIA interface development and analysis as his Master's thesis project, hence some materials in this section are shared between our theses.

User interface evaluation requires the subjects not to be familiar with the interface so that they could evaluate its effectiveness. Measuring the effectiveness of critical point requires the subjects to be already familiar with the interface, so as to reduce bias associated with the learning curve. And this requires measuring how long it takes to perform the same mapping task in two modes: a) using critical points, and b) without using critical points.

The experiment is divided into two sections: a) evaluating the interface, and b) evaluating the effectiveness of using critical points. The first serves as a training session for the second.

III.G.1 Approach

Evaluating the interface The premise of this part of the experiment is to ask a sample group of subjects to perform representative tasks (of varying difficulty) using the SCIA graphical user interface. If a user has any questions or concerns during this portion of the experiment, then these concerns are recorded. This section of the experiment is required to be conducted before the second section to ensure that the subjects are already familiar with the interface. There is also

a short pre-test questionnaire for subjects to fill out, in order to learn about their background and evenly distribute them into two groups for the second part of the experiment.

Evaluating the benefit of using critical points The second part of the experiment is designed to evaluate how much effort using critical points saves when performing schema mapping tasks. Half subjects are asked to run two new representative tasks (timing task A and B) using critical points, while the rest are asked to perform the same task without using critical points. The time it takes each subject to create a correct mapping is recorded, starting from click on the Match button and ending at when the subject expresses satisfaction with the translated target instance.

III.G.2 Recruiting and Grouping Subjects

The experiment is designed to collect input from 20 subjects to reduce bias linked to personal background, abilities and experience. The subject pool includes undergraduate and graduate students in computer science and graduate students from other disciplines. The subjects should be familiar with the basic concepts of databases, structural representation of data, and XML. But they should not have profound knowledge about schema mapping and data transformation, otherwise their knowledge may distract them from the experiment by raising too many questions about how tool works internally. Most of our subjects have taken some database classes, but not working on schema mapping projects. The subjects should also not be family or friends of the people conducting the experiment because they might not give honest answers.

The 20 subjects are divided into 2 balanced groups for the second part of the experiment according to their feedback to the 6 questions about their background in the pre-test questionnaire. The questions include ranking their familiarity to 1) computer science, 2) schemas, 3) databases, 4) schema matching, 5) schema mapping, data translation and integration, and 6) SCIA, with 1 meaning

no experience at all and 5 meaning very experienced. The individual experience rank, group average rank for each question, and total score for each subject are shown in Table III.2 for Group I and Table III.3 for Group II. From the above two tables, we see that the two groups' average ranks for every question are very close to each other, and the average total score (16.7) of Group I is the almost the same as that (16.8) of Group II. We run t-tests over the total score of Group I and Group II, the 2 tails t-test value is 0.97, which indicates the two groups are from the same distribution.

Table III.2 Group I subjects profiles

Subject	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*Avg score*
Computer science	5	3	5	4	1	5	4	4	5	5	4.1
Schemas	5	3	5	3	2	4	1	4	1	2	3
Databases	5	4	5	3	2	5	2	4	2	3	3.5
Schema matching	2	4	1	2	1	4	1	2	1	3	2.1
Data translation	3	4	5	1	1	5	1	3	1	2	2.6
SCIA	2	1	2	1	1	1	1	1	1	3	1.4
Total score	22	19	23	14	8	24	10	18	11	18	16.7

Table III.3 Group II subjects profiles

Subject	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*Avg score*
Computer science	4	5	4	5	4	4	4	4	5	4	4.3
Schemas	4	1	3	5	3	2	4	4	2	2	3
Databases	4	1	4	5	4	2	4	4	5	2	3.5
Schema matching	5	1	2	5	3	1	2	3	1	1	2.4
Data translation	4	1	2	5	2	1	2	4	1	1	2.3
SCIA	1	1	1	3	2	1	1	1	1	1	1.3
Total Score	22	10	16	28	18	11	17	20	15	11	16.8

III.G.3 Selecting Use Cases

To make sure the experiment is doable within a reasonable time frame and to maximumly reduce the impact of subjects' backgound, we choose sample schemas and data from bibliography. These schemas and relationships among them

are straightforward, and no profound domain knowledge is needed for understanding the following mapping tasks in each session. The schemas involved are included in Appendix A.

1. For the training session

   - Task A: $book4 \rightarrow book1$ involving features:
     - Skolem function for generating book id,
     - unit conversion for book price from cents to dollars,
     - string splitting for author and editor full names to separate first name and last name (using user defined functions firstStr and secondStr).
   - Task B: $books\_author2 \rightarrow books\_book3$ involoving features:
     - local n:1 from author's first name and last name to writer's name,
     - unit conversion for book price from USA dollars to Chinese Yuan,
     - regrouping books from grouped by author to grouped by book id,
     - specify constant value for book genre.

2. For the GUI evaluation session

   We ask subjects to run the following tasks using critical points only, since without critical points does not involve much interaction through GUI).

   - Task A: $book2 \rightarrow book3$ involving features:
     - local n:1 for separate first name and last name to full name
     - Skolem function for book id,
     - unit conversion for book price from USA dollars to Chinese Yuan,
     - specify constant value for book genre.
   - Task B: $books\_book3 \rightarrow books\_author$ involving features:
     - regrouping books from grouped by book id to grouped by author.

3. For the timing session

   Each subject from Group I run each of the following tasks using critical points, while each subject from Group II run the same tasks without using critical points.

   - Task A: $book4 \rightarrow book5$ involving features:
     - Skolem function for book id,
     - unit conversion for book price from cents to dollars,
     - string splitting for author's full name to separate first name and last name.
   - Task B: $books\_book \rightarrow books\_author$ involving features:
     - local n:1, from separate first name and last name to full name,
     - regrouping books from grouped by book id to grouped by author.

III.G.4 Fine Tuning

We ran two subjects as a shakedown of the experiment on January 23rd, 2006, 2 days before beginning the experiment. This was to make sure the experiment works well, and the scripts worked well. The test procedures were refined according to the results of the practice tests.

III.G.5 Experimental Setup

The experiment were conducted on a PC running Microsoft Windows XP with Intel Pentium 4 CPU 2.40 GHZ and 512 MB of RAM, from January 28 to February 1, 2006. The subject sat in front of the computer while he/she conducted the test. The facilitator sat beside the subject in order to converse easily. The observer is out of the line of site of the user, but could clearly see the user and the computer screen, taking care of video taping both the subject and the screen. The experiment starts with SCIA already started and ready to be used by the subject.

The layout of the experiment is described in Appendix B. The quantitative results and comments from the subjects are reported in the following sections.

III.G.6 Results of Measuring the Effectiveness of Critical Points

Here we summarize the timing results to quantify how much help using critical points offers. Figure III.26 shows the time that each subject in group I takes to accomplish task A with critical points, and also shows the time that each subject in group II without critical points takes to acomplish the same task; Figure III.27 shows the timing results for task B.

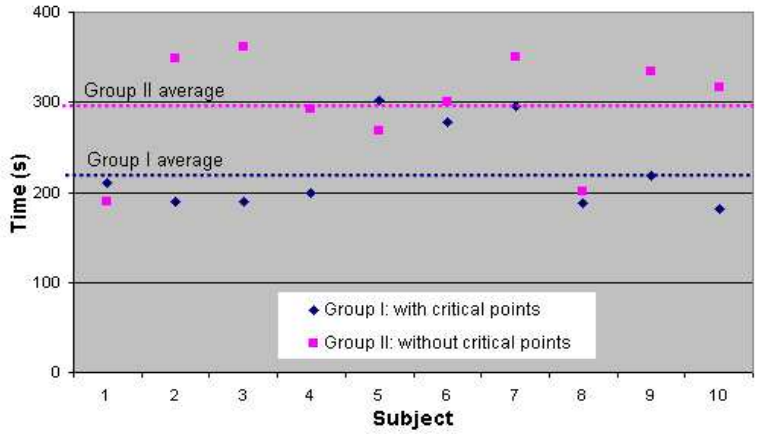

Figure III.26 Time taken for task A

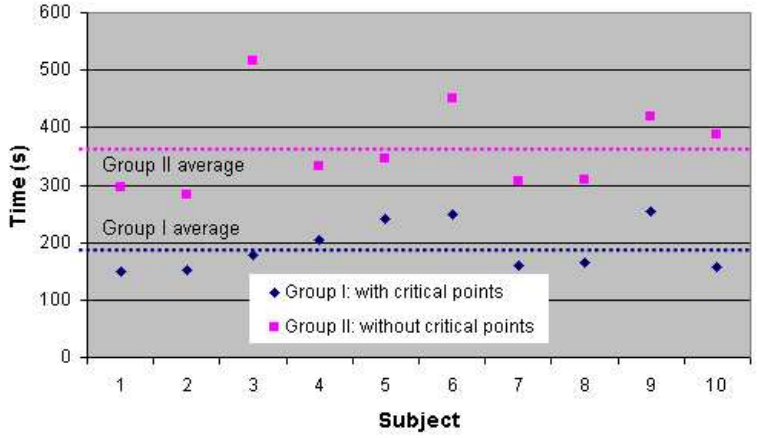

Figure III.27 Time taken for task B

Table III.4 Statistics of the timing results

Measure	*Average Time (s)*		*Standard Deviation*		*T-Test*	
Group/Type	*w/ cp*	*w/o cp*	*w/ cp*	*w/o cp*	*1 tail*	*2 tails*
Task A	225	296	47.76	60.46	0.00486	0.00359
Task B	191	364	42.70	76.77	0.00001	0.00005

We compute the group average time taken for each task with critical points and without critical points, shown as dashed lines in the Figures III.26 and III.27. The group average value and standard deviation for each task are given in Table III.4. The level of significance of the difference between the time with critical points and the time without using critical points was computed using the t-tests shown in Table III.4; this amounts to the probability that the two samples come from the same distribution. For task A, the t-test values are 0.00486 for 1 tail and 0.00359 for 2 tails; for task B, the t-test values are 0.00001 for 1 tail and 0.00005 for 2 tails. The t-test results indicate that two samples are sharply different from each other, more specifically, meaning that for task A, less than five times out of a thousand you would find a statistically significant difference between the means even if there was none (i.e., by "chance"), while for task B, it is one out of 100,000. For the simpler task A, on average using critical points saves 24% of the time, while for the relatively more complicated task B, on average using critical points saves 48% of the time. We measured the complexity of a mapping task by the time it takes on average as in Figure III.28. The trivial task is to map two empty schemas, which requires no work, hence there is 0% saving. Figure III.28 shows the more complex the mapping task is, the more percentage of time using critical points would help save. The curve is a non-linear function, and it looks like it may be exponential. From these obeservations, we hypothesize that the time saved by using critical points grows exponentially with complexity of mapping task.

We also notice that using critical points, different subjects with different background use similar time to finish the same task, especially for the relatively complicated task B, while without using critical points the time taken to finish the

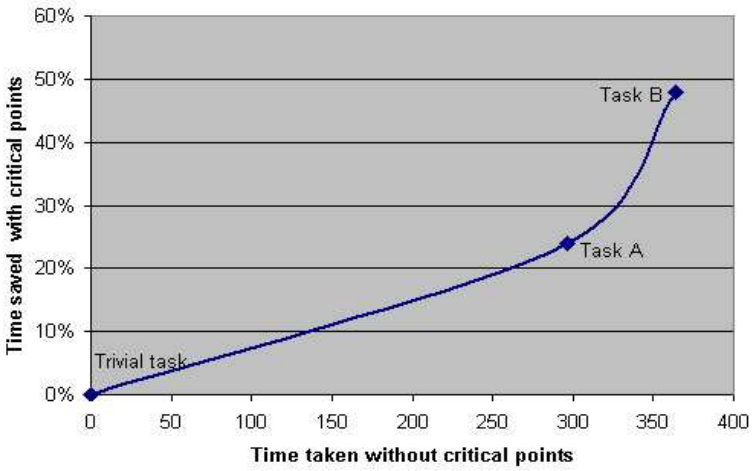

Figure III.28 Percentage time saved by using critical points for tasks with different complexity

same task varies significantly among different subjects, which is confirmed by the significant difference between the standard deviation values for each task in Table III.4. For example, as Figure III.27 shows, without using critical points, for task B, some subjects finish within 5 minutes, but some take about 9 minutes. However, for the same task B, using critical points, all subjects finish within 3 to 4 minutes.

Figure III.29 and III.31 shows the different time taken by the same subject with critical points first and then without for task A and task B respectively. Figure III.30 and III.32 shows the different time taken by the same subject without critical points first and then with for task A and task B respectively.

Figure III.31 and III.32 tell us that using critical points always saves time for every subject for the same task B no matter the order of performing the two tasks. As expected, Figures III.31 and III.32 show that using critical points saves less time for the same task using critical points first and then without critical points than the opposite order. This is due to help from the experience with performing the same task for the second time. From task A, Figure III.29 shows two exceptions, since two subjects take less time without using critical points the second time than with critical points at first. This may be because it is easier to

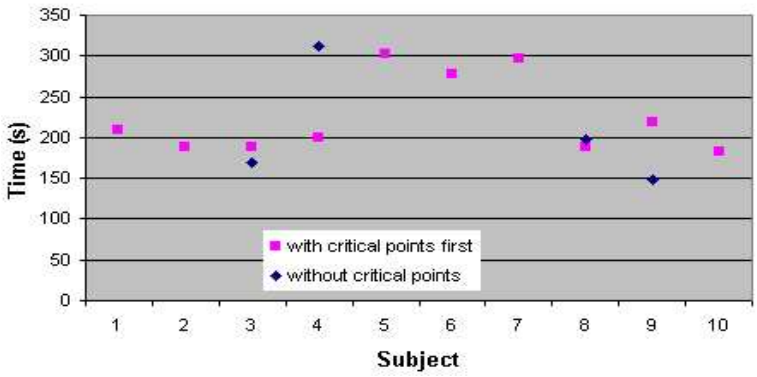

Figure III.29 Time for task A by same subject first with and then without critical points

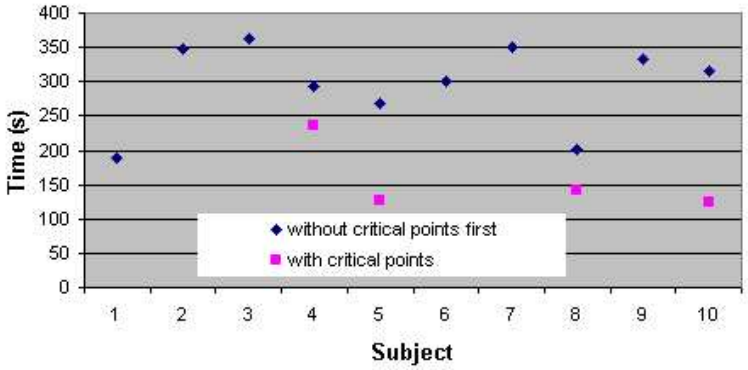

Figure III.30 Time for task A by same subject first without and then with critical points

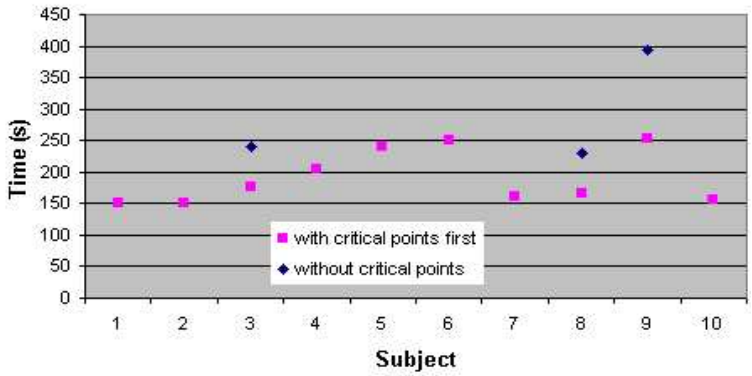

Figure III.31 Time for task B by same subject first with and then without critical points

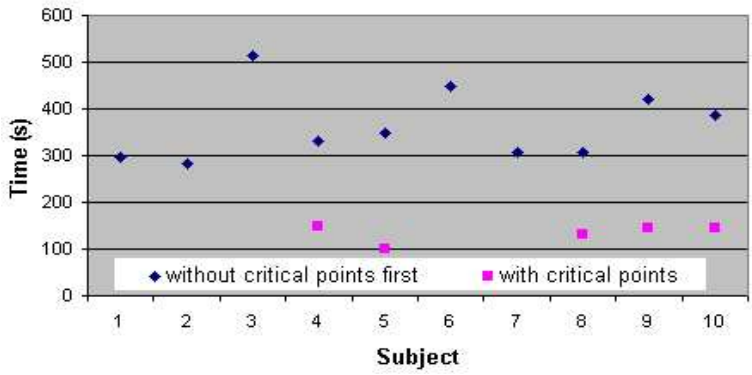

Figure III.32 Time for task B by same subject first without and then with critical points

remember the simpler task and the significant help from the knowledge about the simpler task gained from the first time running eliminate the effect of using critical points.

III.G.7 Interface Evaluation Comments

All 20 experiments were completed within the expected time frame. And all subjects did great job on thinking aloud as we requested and provided very thoughtful comments and suggestions for improving the graphical user interface. In this section, we report what we learn from the user interface evaluation.

- **Mapping Direction** The subjects seemed to be somewhat unsure about the direction of the matches, whether they go from left to right or vise versa. SCIA puts the source on the left and target on the right. It seemed natural and it is also the way the other schema matching tools lay out their interfaces. But many subjects attempted to select the right node first, and then select the left. This might be caused by the interaction with SCIA. The SCIA mapping algorithm attempts to find matching nodes for every node in the target schema. When the users are prompted to resolve a critical point, this is done for a particular target node, prompting the users to attempt to select the right (target) node first. Idealistically it might seem that the users should

be given an option to make matches in any direction, to avoid this confusion. However, there are several reasons why this should not be done. The first is simply consistency - the way to enter a new match should be the same in different stages. The second reason is semantic - the matching should follow the flow of data, from the source schema to the target.

- **Source Data** Quite a few subjects commented on the absence of example data in the interface brought. The current SCIA interface only shows the two schemas, while it is often critical to receive semantic information from the actual data files, because SCIA uses critical points, pays special attention to the development and improvement of the graphical user interface, and provides the extensive mapping options to the users. These options include union matches, local multiple matches, conditions and operations on the matches, constant values and unary functions on the target nodes (such as Skolem functions), and others. For users to take advantage of this functionality, they need to acquire more knowledge about the data than could be seen from the schemas alone. For example, one of the functions the users had to enter dealt with currency conversion, the price in a source schema is in cents, which needed to be converted into dollars by dividing it by 100. Several subjects suggested that the example data should be included into the interface. The main reason that we are not having source data in the interface is semiotic. If the examples are shown as an inlay in the schemas themselves, samples would have to be extracted and re-ordered from their natural layout to the layout of the schemas. This does not preserve the correct relationships between signs in the morphism from their original system to their representation system, thus confusing users. In addition, this would make the display cluttered. If, on the other hand, the example data is shown separately from the schemas, such as in another window, the linking between the two windows would be non-trivial, and instead of easing users' job, it could complicate it. As a matter of fact, users should understand the

source data and both schemas in order to map them correctly, so displaying the data and helping users understand their data are not SCIA's primary functionality.

- **Thinking by Clicking** Some of the subjects "think by clicking", where they tend to highlight nodes while they are making the decision whether or not to use it for a match. Since SCIA is making a match as soon as the subject clicks on the source node and then on the target node, this introduced many unwanted matches. The subjects were then required to delete those matches, which exposed several deficiencies in both the GUI and the matching algorithm. One solution is to always warn the user before a new match is being made, which of course would slow down the mapping process, and might irritate the users. A fine balance must be made to determine how responsive the interface should be to prevent unwanted matches without significant slowdown of the process. One possible solution for preventing the users from "thinking by clicking" is to highlight the target nodes for them. This of course could only be done during semi-automatic matching when the users are prompted to resolve a critical point for a particular node. Highlighting the target node might prevent the users from highlighting it themselves while they try to figure out which source node to choose for the mapping. It would also help to have an "undo" option for a match, which allows easy deleting of unwanted matches or functions on the nodes.

- **Dialog Placement** Currently, dialog prompts are displayed in a separate window in the lower-right corner of the screen. Many subjects thought these dialogs should be placed in the center of the screen, or on the bottom of the main GUI, even a cuple of subjects always moved the dialogs to the center of screen and then read them. But placing the dialogs in the center of the screen would obscure the actual schemas and mapping area, which the subjects might need to use while answering the prompts. For example, the

users might be prompted to enter a manual match if no good prediction could be made, or to enter a function on a mapping with low similarity between the nodes, and then to resume the mapping process by clicking an the "OK" button on the prompt. Placing the dialog on the bottom of the main GUI limits its viewing space. Unless the area is constantly resized, which might obscure the lower tree nodes, some of the prompts might not fit in the allowed area (the size of the prompt depends on the number of options and on the lengths of the node paths, among other things). In this case, scrolling would be required, which is an unintended consequence. Arguably, placing the prompts in the lower-right corner of the screen, the farthest from the main GUI, may be the best solution in order to avoid obscuring the mapping area.

- **Match Line Marks** A valuable suggestion made by several subjects is to mark differently the matches (represented with lines) which have been verified, the matches which are being checked, and the matches which are to be checked. A design challenge for this is overloading of the line representations. Currently, different colors are used for selected and un-selected mappings. Different thickness is used for multiple mappings between the same nodes (differing by condition placed on them). Introducing more colors would further complicate the interface. From the semiotic perspective, we want to preserve the correct relationships between steps in the morphism between the context check algorithm sign system and its representation. As the nodes are being checked one by one, we want to show that they are being checked one by one. Marking the verified nodes is one way to do this.

- **Complex Mappings** A few subjects suggested that SCIA should show the complex mapping formula while it is being entered, possibly in another window, so that the user is warned if he/she is making a mistake and needs to redo the mapping. SCIA supports complex mappings which require entering multiple source nodes, such as for condition or grouping, and entering multi-

ple operations connecting the data contained in these nodes. Although this did not happen frequently during the experiment, users might potentially get confused and enter incorrect information. Without the mapping formula display, the user might not find their mistake until much later, when they generate a view for the mapping. If the schemas are sufficiently large, the users might have a difficulty finding the place where they made the mistake just by looking at the view or the view execution output, so it is logical to warn them as soon as possible. A complex mapping formula is a single entity, where the nodes comprising it are linked by operations. According to algebraic semiotics, operation preservation is one of the optimality principles. Currently, SCIA asks the user to enter the nodes one-by-one prompted by a series of windows to enter individual nodes and operations. The operations are not preserved in the formula representation, thus reducing its quality. The undo mechanism is useful here as it allows the users to correct their entries without having to restart entering the complex mappings. Another idea brought up during a subsequent discussion is to allow users to edit the mapping formula directly in the display. When mappings are entered, they are first stored in several internal structures, for easy access by the algorithms, and then shown as a single string to the user. Allowing the user to modify the mappings by altering their string representation reverses this order, and introduces technical difficulty with possible disambiguation of freehand user input. While the users enter the nodes by selecting them from the trees on the display, they have to enter operations manually. Currently, if such an operation is entered incorrectly, the users are not aware of the problem until the view execution step when the Kweelt engine produces an error. SCIA should attempt to warn the users ahead of time if they had entered incorrect operations. Moreover, it is desirable to provide more intuitive and manageable ways for users to handle complex matches. For example, for a minimal n-to-1 match, currently, notifying the start point and the end point at the

source side and the target side respectively is through pressing the ALT key in one way that works but it is not good visually. In another way, clicking the source part of the big arrow will highlight the source part and means starting to work on the source part, all the nodes selected before next click on the source part form the source component in a minimal n-to-1 match. Clicking the connection part will highlight itself and the match lines involved and notify the system to get ready to visualize or help users formulate the match between the relevant nodes. We thought it was more intuitive and simple. However, some subjects were confused, and they needed explanation multiple times. As discussed in Section II.A, there are many complex matches beside simple 1-to-1 matches, therefore it is too hard to create a mechanism to treat all cases uniformly. For example, SCIA expects global 2-to-1 matches to be treated as 2 separate local 1-to-1 matches, but always treats local 2-to-1 as minimal 2-to-1. It seems that brief and specific instruction should be prompted to users when they are going to enter complex matches. Furthermore, it is also a remaining issue to provide a relatively general graphical interface to formulate complex local n-to-m matches by asking users the involved functions, conditions and how to combine them together with the relevant paths.

- **Explicit View Generation** SCIA contains "Generate" and "Execute" view options. While the terms seem to be logical, many subjects were unsure what these terms meant and in which order these two options should be executed. The current sequence is strict in that the users must generate a view before executing it. While the generated view is the primary result of schema mapping algorithm, many subjects were not familiar with the Quilt syntax, and were inclined to skip examining the generated view and just look at the translated data. This is also the step when many of the semantic errors in the mapping become apparent. For example, the price conversion errors would only be seen when looking at the data file. Some subjects opted to

skip the explicit view generation step altogether, i.e., they just wanted to be able to execute a view, which would generate it as an intermediary step, then execute it over the sample data. There are several problems with this. First, while view generation is the required step in the schema mapping process, the view execution step is optional, and some users might not do it, especially if they do not have sample data. Second, the generated views can be manually edited, and be saved and reused separately from the transformed data. The data transformation step is a separate process from schema mapping altogether, and should not be emphasized in a schema mapping tool interface. On the other hand, SCIA is really intended to eventually be a data integration tool.

- **Schema Validation** Some subjects tried to generate a view and execute it before the system acquired all the necessary input, so the translated data might not fully adhere to the user requirements. Some subjects suggested that SCIA would benefit from providing the option to validate the resulting XML data file against the target schema. While this might potentially find errors, it would not find any semantic mistakes, such as the correctness of entered functions, conditions, and multiple matches. In our experiments, all errors in the generated views, which prompted the users to revisit their mappings, were semantic in nature, not structural. This class of errors would not be found by a simple schema validation step, so the value of schema validation here is questionable.

- **Prompt Organization** Effort is needed to provide more informative and consistent feedback at critical points during the mapping process. The information and questions should be categorized appropriately, organized consistently for each category and attached with clear explanation to help users understand what is going on and how to provide input that the system can understand. Particularly, when a n-to-m match happens, the terminologies

about union and local multiple matches, and how to edit them correctly should be explained by using examples and screenshots.

These interface comments showed the overall high quality of the SCIA interface, and valuable in showing ways to improve the interface. They also showed that the class of subjects used was not familiar with all the necessary concepts about schema mapping, which is exactly to be expected given the way that they were recruited.

III.H Discussion

Our work on schema mapping has been motivated by our experience in eco-environmental data management and analysis that often require semantic mappings for transforming from one schema to another and integrating data from different sources. We started by simply combining existing schema matching algorithms together to see how well they perform for some real world schemas. To save the total user effort for creating correct schema mappings, we came up with the idea of critical points, developed an algorithm for detecting and resolving critical points and provided a graphical user interface for efficient user interactions. In the following we discuss our development methodology, future work on view generation, and applications to workflow systems and experiments.

III.H.1 Development Methodology

SCIA differs from other related tools and many software systems, because its implementation and improvement has been driven by examples, without any theory for schemas and schema mapping behind it at first. Later on, initiated by its promising results, Goguen worked out the algebraic semantics for schemas and schema morphisms [40, 34]. We then upgraded the implementation based on the abstract schemas and schema morphisms. There is much more that could be done in this direction.

III.H.2 Integration with Workflow System Kepler

The preliminary integration of SCIA with Kepler shows that SCIA's matching part can be used for suggesting connection between workflow components, which should be more meaningful for connecting many more actors and more complicated actors than those discussed in section III.E. However, it will be much more interesting to explore the full functionality of SCIA from matching to view generation for suggesting necessary transformation steps between components when structural or semantic heterogeneities exist as in the various mapping cases. Kepler is built on top of Ptolemy, and hopefully Ptolemy will exploit XML Schema as its component input and output specification, so that SCIA can generate data transformation scripts between actors. Otherwise, SCIA needs to work with the existing type system of Ptolemy, which would require significant changes to the implementation of SCIA.

III.H.3 Experiments with Large Mapping Tasks

The formal experiments that have been conducted are with small common sense tasks. They are made up based on the W3C use cases for XQuery, and they involve interesting complicated mapping cases, including nodes without matches but having constant values or requiring unique value for each instance, local n-to-1 matches, conditions, functions and grouping. Therefore they are very good choices for experiments, representive and doable for general subjects within reasonable time. But people may argue that their size is not representative of the real world applications. As a matter of fact, the workflow component interface structures we met in scientific workflows in Kepler and the typical message schemas in e-commerce we have seen are usually smaller than these bibliographical schemas, while the heterogeneities are similar. Therefore the experimental results with these small tasks are valuable.

However, for data integration and data transformation in general, the average size of schemas is much bigger than these example schemas for books.

We have tested SCIA over many large complex schemas from ecology, biology and business on our own, which are too hard for ordinary subjects to perform within reasonable time. In the future we may recruit some researchers interested in data transformation and integration in these domains to do experiments with much larger and more complex schemas from real applications. In particular, we would like to confirm hypothesis about exponential growth of time saved with complexity of task.

III.H.4 Formal Experiments with Research Prototypes

Our experience shows that formal experiments are effective on academic prototype applications especially when the quality of the interface has a direct relation to the usability of the tool itself and the algorithms behind it. Such experiments are routinely conducted in the industry, but are often avoided by academics, although there are many research projects that would benefit from this experience. The main reason for avoiding these experiments is the belief that prototype applications would face criticism unrelated to their main functionality, mostly linked to software bugs or missing options. SCIA is an example-driven prototypes, which is designed to work primarily with limited input data. Yet, the formal experiment proved that implementation deficiencies did not obstruct users' view of the application. Instead, the users were able to grasp the key new concepts that SCIA provides. Resolving critical points was shown to significantly reduce the time it takes to make a correct mapping. This is only possible if the user interface allows for it. A poor interface would slow down the mapping process and compromise the overall results. The subjects did not seem to be distracted with the bugs or cosmetic deficiencies discovered in the tool, but instead concentrated on the mapping tasks.

III.I Summary

To save the total user effort for creating correct semantic mapping between schemas, we introduced the idea of critical points for interactive schema mapping and implemented it in our schema mapping tool SCIA. SCIA differs from related systems with which we are familiar in several aspects: (1) Formal experiments show it significantly reduces total user effort by using path contexts in combination with multiple matching algorithms to find critical points where user feedback is necessary and maximally helpful, and asking users specific questions with adequate information; (2) it handles the n-to-m matches that often exist in practice, but are usually ignored in other systems; (3) it also handles matches involving semantic functions and/or conditions; (4) it outputs mappings in both path/concept correspondence format and executable view format to support transformation of data from the source schema to target schema and answering queries over target schema; (5) it has rigorous semantic models for heterogeneous schemas and schema mappings, based on algebra [34, 40].

Chapter IV

Schema Mapping Tool Interface Analysis and Design Principles

Schema mapping is a critical step for many important database applications. But because it cannot be fully automated, user input is necessary to generate correct mappings. Moreover, the users of schema mapping tools are often domain scientists rather than computer scientists. Hence these tools should have a simple graphical user interface that captures the essential structure of schema mapping tasks, and that helps users understand the tool and their tasks. For these reason, a number of schema mapping tools now have GUIs, or are developing them. Unfortunately, systematic, effective principles for the design of schema mapping GUIs are lacking. There are two main reasons why developing such principles has been difficult: (1) user interface design has no widely accepted scientific basis; and (2) schema mapping tools are based on different (usually implicit) notions of schemas and mapping, and are used for different tasks.

We address the first problem using the emerging methodology of *algebraic semiotics* [30], and address the second problem by retrospectively constructing *minimal models* for various tools. The minimal model is a model of the basic structure and functionality of a system, plus user values. If constructed correctly, a minimal model corresponds to what a class of tools can do, and hence provides a

basis for user interface design. Algebraic Semiotics, which is introduced in Section IV.A, provides a rigorous method for user interface design, using *semiotic morphisms* to formalize the intuitive notion of representation [30], and providing criteria for comparing the quality of representations based on how well they preserve structure; we use it to analyze the interfaces of Rondo [63], Clio [46, 64, 81, 80], COMA/COMA++ [17, 4] and our own SCIA, to suggest improvements, and to summarize interface design principles specific for schema mapping tools. Sections of this chapter summarize each system and give its minimal model, , i.e., a simplified semiotic theory of the system, before analyzing its interface semiotically. This is joint work with Goguen and Zavelov, some ideas and results are described in an ongoing paper draft and a report co-authored with Goguen and Zavelov [95, 41]

IV.A Algebraic Semiotics and User Interface Design

There are many useful general principles for user interface design (UID) and human-computer interaction (HCI). But they cannot take sufficient account of the specific requirements of specific system users. Semiotics is the study of signs. An early insight of semiotics is that communication is always mediated by signs [30]. Because UID can be seen as representing the underlying functionality of a system, by using signs in the interface, it seems natural to study user interface design using semiotics. Classical semiotics (see [13] for an overview) originated in work of the American logician Charles Sanders Peirce [78] and the Swiss linguist Ferdinand de Saussure [85]. Peirce sought to understand how particular signs convey their meanings, while Saussure emphasized the importance of viewing signs as compound entities that participate in families of other signs. Classical semiotics did not develop in a sufficiently rigorous way to support complex engineering applications, nor did it explicitly addresses the representation of complex signs, or signs that change over time, each of which is needed for UID.

The research on algebraic semiotics [30] attempts to make this area more

systematic, rigorous, and applicable, as well as to do justice to its social and cognitive foundations. It combines aspects of algebraic specification and social semiotics. By generalizing from classical algebra to hidden algebra, it also provides a way to handle dynamic interfaces [33]. It has been applied to user interface design, and other areas including information visualization, the representation of mathematical proofs, multimedia narrative, virtual worlds, and metaphor generation [30, 37, 38].

Algebraic semiotics formalizes sign systems as algebraic theories with additional structure that captures the values of the social group interested in the sign system. The essential idea of algebraic semiotics is that representations are morphisms from one sign system to another, and more important, the design and analysis of such morphisms are not only technical but also social. The social and value issues determine the ordering of a complex sign system. A user interface for a computer system can be seen as a semiotic morphism from the underlying abstract machine representing what the system can do to a sign system of windows, buttons, etc. [30]. Criteria are provided for comparing different semiotic morphisms and for aiding semiotic morphism design based on structure preservation.

IV.A.1 Sign Systems

In algebraic semiotics, sign systems are described by semiotic theories which capture the systematic structure of complex signs. Most signs are composed by other signs, and hence sign systems usually have a classification of signs into certain **sorts**, and some rules for combining signs of appropriate sorts to get a new sign of another sort. Sorts may have a hierarchical structure, for example, sort `ARTICLE` is a subsort of `PUBLICATION`. The hierarchy of complex signs is expressed by the levels of sorts. The rules for composing signs are called **constructors** in a sign system. Constructors are essentially functions that build new signs from given signs and possibly additional **parameters**. For example, a "cat" sign on a computer screen may have parameters for the size and location. Some constructors may be more important than others, giving rise to a **priority** ordering of the

constructors for signs of a given sort. A primary constructor that constructs the most important sign of a sort has the greatest priority. Priority is a partial ordering, since one constructor is not necessarily more important than another. A sign system is defined formally as follows in [30].

Definition 1: A **sign system** S consists of:

1. a set S of **sorts** for signs;
2. a partial ordering on S, called the **subsort** relation and denoted $\leq$;
3. a set V of **data sorts**, for information about signs, such as colors, locations, and truth values;
4. a partial ordering of sorts by **level**, such that data sorts are lower than sign sorts, and such that there is a unique sort of maximal level, called the **top sort**;
5. a set C_n of level n **constructors** used to build level n signs from signs at levels n or less, and written r: $s_1...s_k d_1...d_l \rightarrow s$, indicating that its ith argument must have sort s_i, its jth parameter data sort d_j, and its result sort is s; constants $c :\rightarrow s$ are also allowed;
6. a **priority** partial ordering on each C_n;
7. some relations and functions on signs; and
8. a set A of sentences (in the sense of logic), called **axioms**, that constrain the possible signs.

IV.A.2 Semiotic Morphisms

Semiotic morphism provides a way to describe the translation of signs in one system to signs in another system. In user interface design, generating a good icon, file name, explanation, or arranging text and graphics together in an

appropriate way, all involve moving signs from a source sign system to a target sign system.

Given a source and a target sign system, a semiotic morphism from the source to the target consists of partial functions from sorts in the source to the sorts in the target, functions from the constructors of the source to the constructors of the target, and functions from the predicates and functions of the source to the predicates and functions of the target[30]. The formal definition is as follows [30].

Definition 1: Given sign system S_1 and S_2, a **semiotic morphism** $M : S_1 \rightarrow S_2$ from S_1 to S_2, consists of the following functions (all denoted M):

1. sorts of $S_1 \rightarrow$ sorts of S_2,
2. constructors of $S_1 \rightarrow$ constructors of S_2, and
3. predicates and functions of $S_1 \rightarrow$ predicates and functions of S_2,

such that

1. if $s \leq s'$ then $M(s) \leq M(s')$,
2. if $c : s_1...s_k \rightarrow s$ is a constructor (or function) of S_1, then (if defined) $M(c)$: $M(s_1)...M(s_k) \rightarrow M(s)$ is a constructor (or function) of S_2,
3. if $p : s_1...s_k$ is a predicate of S_1, then (if defined) $M(p)$:$M(s_1)...M(s_k)$ is a predicate of S_2, and
4. M is the identity on all sorts and opertations for data in S_1.

IV.A.3 Quality of Semiotic Morphisms

The goal of user interface design is to produce high quality representations. However, it is not clear how to measure the quality of a representation. In addition, design is subject to constraints and typically involves tradeoffs, i.e., comporimises between competing measures of success, such as cost, size, complexity

and response time. Limits on human capability for dealing with complex displays imply that some information may have to be compressed, deleted, or moved elsewhere, hence we need priorities on what should be presearved. The entire structure of the sign systems should be considered in determining what makes one representation better than another. The structure that is preserved by semiotic morphisms provides an important way to compare their quality. A good semiotic morphism should preserve as much of the structure in the source sign system as possible, by mapping sorts to sorts, subsorts to subsorts, data sorts to data sorts, constants to constants, constructors to constructors, and axioms to axioms [30]. In addition to preserving the formal structure of the algebraic theories for the source sign system as much as possible, a good semiotic morphism should also preserve the priorities and levels of the source space, because priorities and levels often embed the values of the social group interested in the system. The extent to which a semiotic morphism preserves the various features of semiotic theories determines its quality [33]. The following definition in [30] gives precise ways to compare the quality of representations.

Definition 3: Given a semiotic morphism $M : S_1 \rightarrow S_2$, then:

1. M is **level preserving** iff the partial ordering on levels is preserved by M, in the sense that if sort s is lower level that s' in S_1, then $M(s)$ has lower or equal level than $M(s')$ in S_2.

2. M is **priority preserving** iff $c < c'$ in S_1 implies $M(c) < M(c')$ in S_2.

3. M is **axiom preserving** iff for each axiom a of S_1, its translation $M(a)$ to S_2 is a logical consequence of the axioms in S_2.

4. Given also $M' : S_1 \rightarrow S_2$, then M' is (at least) **as defined as** M, written $M \subseteq M'$, iff for each constructor c of S_1, $M'(c)$ is defined whenever $M(c)$ is.

5. Given also M': $S_1 \rightarrow S_2$, then M' **preserves all axioms** that M does, written $M \preceq M'$, iff whenever M preserves an axiom, so does M'.

6. Given also $M' : S_1 \rightarrow S_2$, then M' is (at least) as **inclusive** as M iff $M(x) = x$ implies $M'(x) = x$ for each sign x of S_1.

7. Given also $M' : S_1 \rightarrow S_2$, then M' **preserves** (at least) **as much content as** M, written $M \ll M'$, iff M' is as defined as M and M' preserves every selector that M does, where a morphism $M : S_1 \rightarrow S_2$ **preserves** a **selector** f_1 of S_1 iff there is a selector f_2 for S_2 such that for every sign x of S_1 where M is defined, then $f_2(M(x)) = f_1(x)$, where

8. a selector for a sign system S is a function $f : s \rightarrow d$, where s is a sign sort and d is a data sort of S such that there are axioms A' such that adding f and A' to S is consistent and defines a unique value $f(x)$ for each x of sort s. For example, each parameter of a constructor has a corresponding selector to extract its value.

The intuition for the 7th definition is that content is preserved if there is some way to retrieve each data value of the source sign from its image in the target sign system. The above definition provides more rigorous guidelines than the general principles for user interface design [89, 60] and criteria for analyzing user interfaces and comparing different interfaces for the same software system or similar systems.

IV.A.4 Algebraic Semiotic Methods to Interface Design and Analysis

The following are the recommended steps of the semiotic method for user interface *design* [35].

1. Describe the source as a semiotic theory, including constructors and level and priority orderings;

2. Determine the highest priority constructors and selectors for the top level sort;

3. Design a target semiotic system to display this information in the required medium;

4. Design a morphism to preserve the selected structure and omit the rest; if more information is desired, then consider other levels with their highest priority constructors and selectors.

There are always tradeoffs in user interface design since it is a social and engineering problem. Research and experiments show that in general preserving high levels is more important than preserving priorities, and that preserving form is more important than preserving content [31]. The following four principles summarize some significant contributions that algebraic semiotics can make to the design process [35]:

1. Sort preservation: The most important sorts should be preserved, where their importance is given by the level ordering.

2. Constructor preservation: The most important constructors should be preserved, where their importance is given by the priority orderings.

3. Axiom preservation: The most important axioms should be preserved.

4. F/C: When something must be sacrificed, it is preferable to preserve form at the expense of content.

Although Principle F/C is probably the most important, and at this time is certainly the most thoroughly studied and supported, there are other principles that deserve attention, although the range of their applicability has not yet been carefully examined: Principle L/P says it is more important to preserve levels than to preserve priorities; Principle HL/LL says it is more important to preserve higher levels than lower levels; Principle HL/C says it is more important to preserve high levels than content; Principle P/C says it is more important to preserve priorities than content [35].

The following is a recommended semiotic method for *analysis* of an interface [35].

1. Identify as much as is relevant of the social context of the interface and its system, including the goal of the interface, and the nature of the user community;

2. Identify key properties of the source and target semiotic theories, e.g. the major affordances of the source objects;

3. Identify the semiotic morphism involved;

4. Think about what is preserved by the morphism;

5. Consider whether elements of the display are symbolic, indexical, iconic, or diagrammatic iconic, and/or

6. Involve significant sensory-motor (image) schemas, and/or

7. Involve some blending.

Though algebraic semiotics provides us a systematic way to formalize the semiotic spaces and the morphisms between them, in practice, interface designers and analysts can benefit just from applying its principles, such as identifying and preserving key features of the source space, without doing a great deal of formalization [33, 95]. This is what happens in this Chapter. For scientists seeking to understand principles of design and engaged in constructing and testing theories, the algebraic specification language OBJ [39] and BOBJ [32] and their implemented systems can be used to describe the semiotic spaces involved in a detailed formal way, and to test hypotheses with calculations and experiments with users [33].

IV.B Semiotic Approach to Get Schema Mapping Interface Design Principles

We use the methodology of algebraic semiotics to get systematic, effective principles for the design of schema mapping interfaces.

First, we analyze the interfaces of existing schema mapping tools by using the steps recommended for interface analysis described in the above section. The first 4 steps for interface analysis are essential and required, while the last 3 steps can be skipped, hence we go through only the first 4 steps for semiotic analysis of schema mapping tool interfaces.

1. Identify the goals (schema matching, view generation, or both) of the schema mapping system and its interface, and the nature of its user community;

2. Identify key properties of the source theory, i.e., the minimal model for schemas and mappings;

3. Identify the semiotic morphism involved, e.g., how the schemas, mappings and actions are displayed in the interface;

4. Think about what is preserved by the morphism, i.e., the structures of the schemas and mappings.

Second, we identify interface design principles specific for schema mapping tools, by applying the general design principles of algebraic semiotics [35], such as sort preservation, constructor preservation, axiom preservation and F/C, to the minimal models of schema mapping systems understood from the first step.

The source space of a system, which is often dynamic, can be described using hidden algebra in a way both adequate and precise. However, an informal understanding of the functionality and the user values is an adequate basis for analyzing intefaces in algebraic semiotics. Hence the starting point for our analyses of a schema mapping system takes a minimal model based on informal understanding.

IV.C Rondo

We first analyze the schema matching component of Rondo, a generic platform for model management [63]. In the following sections, we discuss its minimal model, analyze its mapping interface, and compare it with the SCIA interface.

IV.D Minimal Model

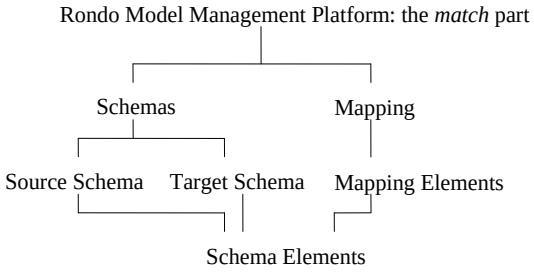

Figure IV.1 Rondo match part: source semiotic space

In Rondo, models including schemas are directed labeled graphs. The nodes of such graphs denote model elements, such as relations and attributes in relational schemas, type definitions in XML schemas, or clauses of SQL statements. Each element is assumed to be uniquely identified by an object identifier (OID). Inter-schema correspondences are called mappings and morphisms in Rondo; these are binary relations, over two sets of OIDs, i.e., a set of pairs (<src-elt>, <tgt-elt>) drawn from OIDxOID, where <src-elt> is an element in the source schema, and <tgt-elt> is an element in the target schema. Notice that such a simple match (<src-elt>, <tgt-elt>) does not say how <src-elt> relates to <tgt-elt>, so it does not carry semantic information about transformation of instances that conform to the schemas (e.g., there are no WHERE clauses). The above overview suggests that the Rondo match part source semiotic space consists of the following main sorts as shown in Figure IV.1:

1. a top sort, which is the entire Rondo match part;
2. 2 second level sorts:
   (a) Schemas, which include a source schema and a target schema; each schema consists of schema elements.
   (b) Mappings, which describe how the source schema and the target schema correspond; each mapping consist of a set of mapping elements;
3. 2 third level sorts:
   (a) Schema elements;
   (b) Mapping elements, including a source element and a target element.

Rondo's source semiotic space also contains some constructors. For example, a primary constructor builds mapping from a pair of schemas.

IV.E Interface Analysis

Now let's identify semiotic morphism from the source space to the target space, i.e., the Rondo interface. Users of Rondo interact with the system by providing scripts first, then through the Mapping Editor GUI shown in Figure IV.2 to edit the generated morphisms that are displayed as red lines between corresponding elements. Hence the primary constructor for mappings is accessible only through the command line, while the mappings and mapping elements are accessible through the GUI.

If a mapping is shown, the user interface is always in one of the two modes: (a) a single element is selected and all of its correspondences are shown, with the best as a red line and the others as grey lines, as shown in Figure IV.3, or (b) no elements are selected and the correspondences between all elements are shown as red lines in Figure IV.2. To switch back and forth between these two modes, users can simply click on the same element multiple times. Once an element

has been selected, the users can hold the SHIFT key and select/unselect matching elements on the other side by using a mouse. Users can delete all incident arcs for an element by pressing the CTRL key and clicking on the element. The numeral on each line represents the confidence of this correspondence in terms of a similarity value from 0% indicating strongly dissimilar to 100% indicating strongly similar.

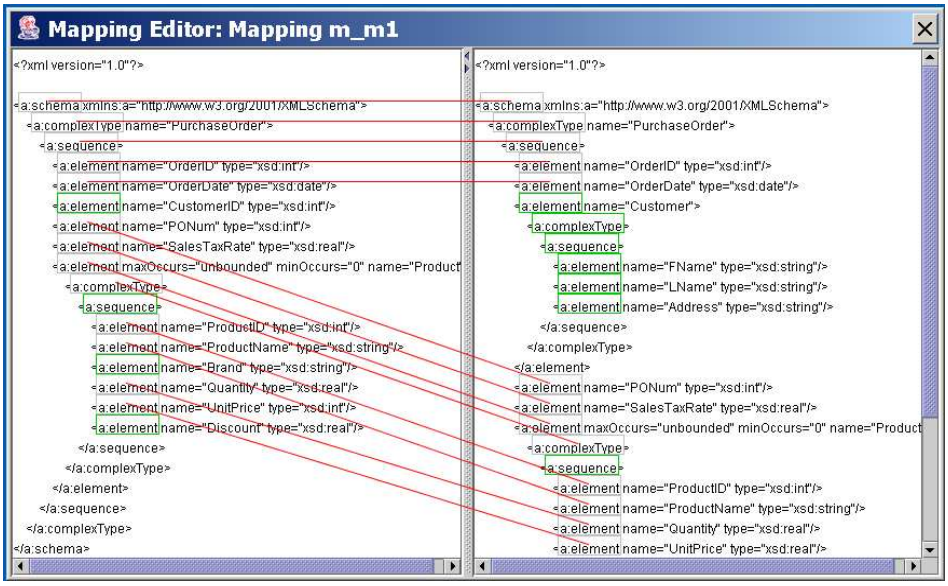

Figure IV.2 Rondo Mapping Editor: showing all correspondences between all elements

Rondo displays the schema files directly in the GUI, with all details shown to the users. This way does not visualize the document structure intuitively. More important, it does not distinguish the shared nodes with different contexts, making it hard to illustrate context-sensitive matches. In our example of matching bibliographical DTDs, `name` → `author/name` does not represent explicitly `source:book/author/name` → `target:book/author/name`, because there are also `publisher/name` and `editor/name` besides `author/name` in the source schema. It is desirable to make data types, cardinality and integrity constraints accessible to users since they are very important for making decisions. SCIA presents these aspects of information in a different way, not simply showing the schema file as Rondo does. SCIA extracts and displays schemas structures in the tree format,

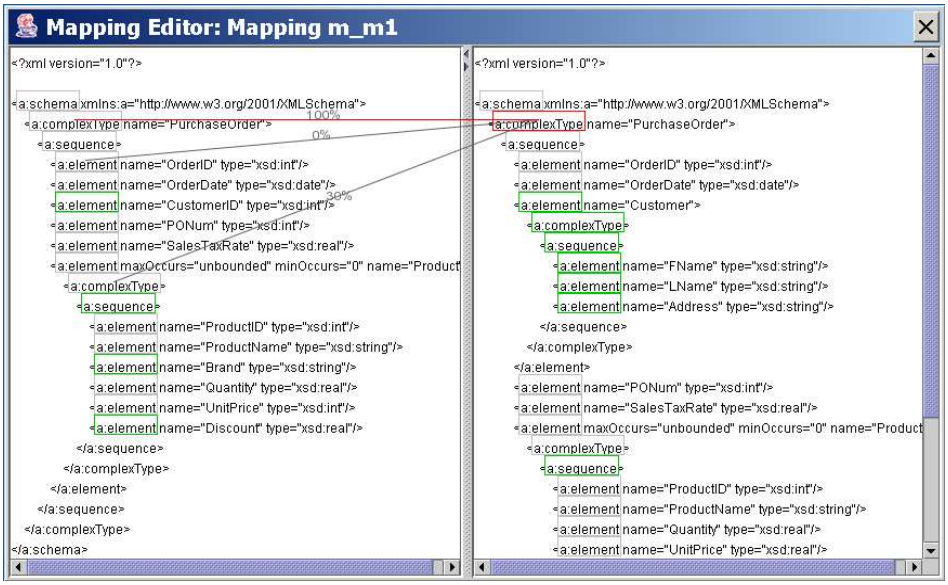

Figure IV.3 Rondo Mapping Editor: showing match candidates of a specific node

and display data types, cardinality and integrity information in its GUI.

Rondo's minimal model contains only simple 1-to-1 matches. These are represented with straight lines from one node to another in its GUI. Rondo Mapping Editor shows top 3 matching candidates for an element. SCIA shows only the best or the user input matches in the static main GUI. Only during context check does SCIA show the top 2 candidates. It is desirable to make multiple top candidates accessible through GUI. In Rondo, a match line for a correspondence is selected by choosing one element first, then holding the SHIFT key and clicking the matching element on the other side, while in SCIA users can simply click the match line in interest when they want to select it, and can view and edit its content with the right-click of a mouse.

IV.F COMA/COMA++

COMA [17] or its extension COMA++ [4] is a schema matching tool, finding only 1-to-1 correspondences. It consists of two major components, an extensible matcher library and a uniform combination scheme for combining results

of matcher executions. Users can specify a match strategy by selecting matchers from the matcher library and the strategies for combining them. The users can also improve the final match results manually through its GUI.

IV.F.1 Minimal Model

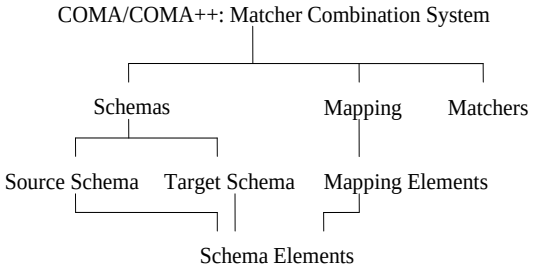

Figure IV.4 COMA/COMA++ source semiotic space

In COMA, schemas are rooted directed acyclic graphs. Schema elements are graph nodes connected by directed links of different types, e.g. for containment and referential relationships. Its output mapping, a binary relation over the two sets of graph nodes, i.e., a set of mapping elements, indicates which elements of the input schemas logically correspond to one another. A mapping element, (<src-elt>, <tgt-elt>), is a pair of a source schema node and a target schema node with a similarity value between 0 and 1 indicating the plausibility of their correspondence. COMA is focused on combining multiple matchers, and it allows users to select which matchers to use explicitly. SCIA on the other hand allows users to select matchers through configuring their coefficients of different matchers. Hence COMA/COMA++'s minimal model, i.e., its source semiotic space, contains not only schemas and mappings but also matchers, shown in Figure IV.4.

IV.F.2 Interface Analysis

Figure IV.5 shows COMA++ GUI and its matching strategy configuration window. Similar to that in SCIA, most of COMA++'s GUI is a white window separated by a vertical bar, where displays the source schema tree on the left and

the target schema tree on the right using the common file system representation, with indentation to represent the containment relationship of elements in schemas. Because a mapping element in the minimal model of COMA and COMA++ consists of only a single source element and a single target element, its GUI contains only match lines from one element on the left to another on the right. Notice that a significant portion of the left side of the COMA++ GUI is used to display the list of schema file names and some details about the currently selected file. It is understandable to have the schema repository in the GUI, because it is more convenient for users to scan the list of available schema files and select existing matching results for reuse. However, other schemas that are not currently being matched are not really a part of the source space of the matching system. They therefor have the lowest priority, not deserving a significant portion in the target space - the main GUI. Some properties of schema elements are displayed on the COMA++ GUI, such as element names and data types. However cardinality information, which is important for making matching decisions, is not accessible.

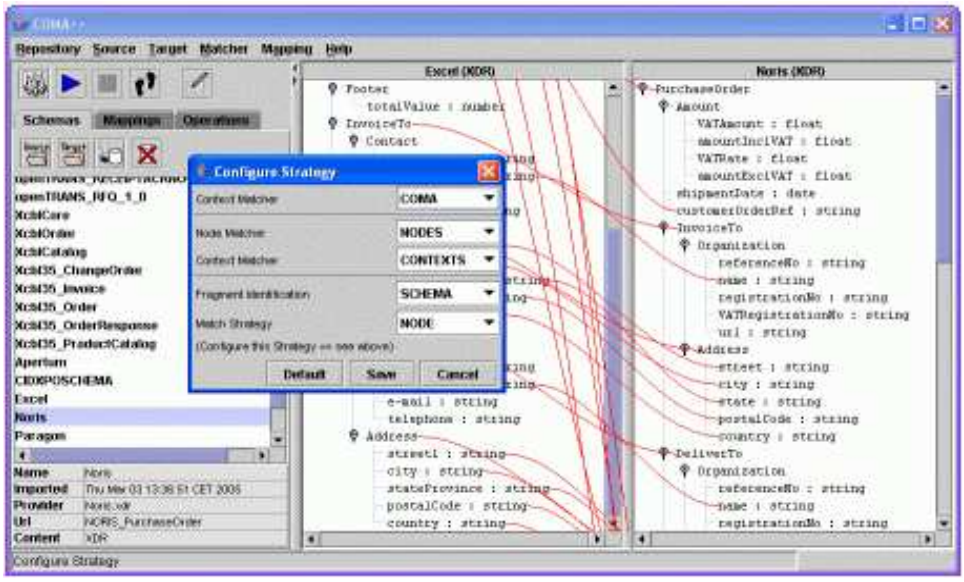

Figure IV.5 COMA++ user interface

IV.G Clio

Clio [46, 64, 81, 80, 47] is a query discovery tool. It introduced an interactive schema-mapping paradigm in which users are relieved from the manual definition of views. From the correspondences provided by a plug-in schema matching tool or by users via its GUI, Clio computes a view for the target schema over the source schema using traditional DBMS optimization techniques, and allows users to verify correctness of the generated view by checking example data translation results.

IV.G.1 Minimal Model

Clio has its own nested relational data model for schemas. It includes a set of atomic types, set types and record types. Within a record, elements are non-repeatable. Repeatable elements are modeled by set types. A set is represented by a set ID and an associated set of children, in order to capture the graph-based data model of XML. A schema is assumed to consist of a single named (root) type. A relational schema with multiple tables is modeled by a record type, the components of which are set types (one for each table).

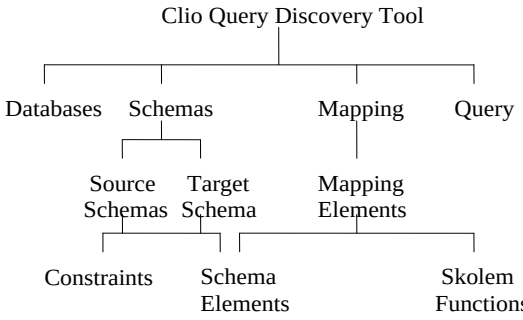

Figure IV.6 Clio source semiotic space

Clio has models for schema mappings at different levels. Inter-Schema correspondences are modeled by element (attribute) correspondence. An element correspondence is a pair of a source element and a target element, (<src-elt>, <tgt-

elt>). One of the reasons for Clio to use this simple element correspondence is that it assumes element correspondences are independent of logical design choices such as grouping of elements into tables or nesting of records or tables (for example, the hierarchical structure of an XML schema). However we don't agree with this assumption; see Section IV.E. Logical Mappings, which are the rpretation of the correspondences faithful to the semantics of the schemas, are modeled by source-to-target-dependencies. Then the source-to-target-dependencies are compiled into low level language-independent data translation rules, taking care of creating new values and grouping of nested elements. Finally these rules are compiled into a transformation language. Clio began by dealing with relational databases. Such database may have multiple tables. Clio regards each table schema as an individual schema. Naturally, it supports mapping multiple source schemas to a target schema. It creates all possible logical mappings, and asks users to verify which is wanted by executing them and reviewing the translated instances. Hence, source databases are part of the tool. These suggest that Clio's minimal model contains the following main sorts, also shown in Figure IV.6:

1. a top sort for the whole system, called Clio;
2. 4 second level sorts:
   (a) Databases, which contain source data;
   (b) Schemas, which include some source schemas and a target schema; each schema consists of schema elements and constraints;
   (c) Mappings, which describe how the source schema and the target schema correspond, and consist of a set of mapping elements; a mapping element contains a source element, a target element and possible Skolem functions for generating values;
   (d) Queries, which can be executed to translate data from the source to the target.

IV.G.2 Interface Analysis

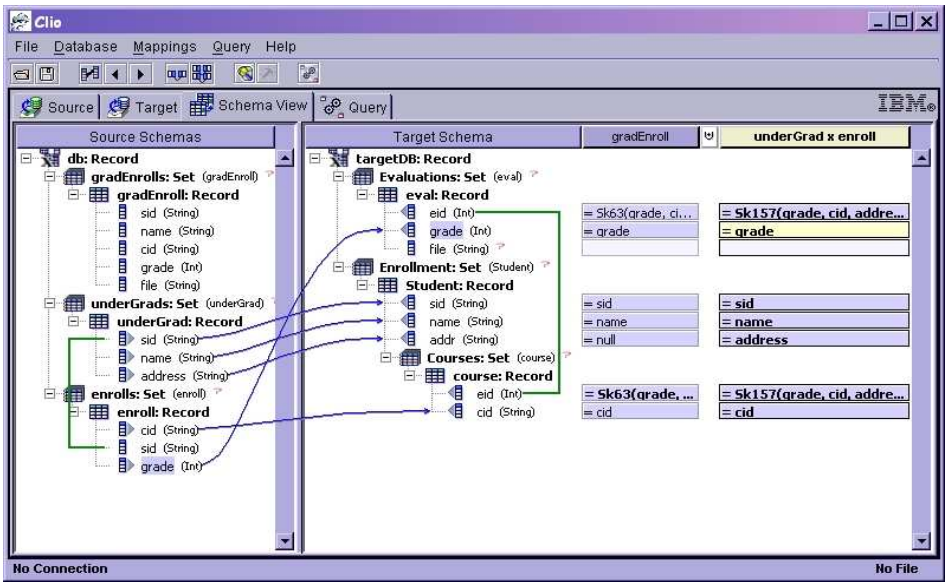

Figure IV.7 Clio: value correspondences and resulting mapping

Figure IV.7 shows the Clio GUI, in which some value correspondences and the resulting mapping are displayed. As in SCIA, most of Clio's GUI is a white window separated by a vertical bar, where displays the source schemas on the left and the target schema on the right using indentation to represent the containment relationship of elements.

Clio focuses on query discovery only, thus there is no matching constructor accessible from the Clio GUI. Clio supports Skolem functions for generating values, but it does not handle local n-to-m matches with functions and conditions. So Clio GUI contains only lines for 1-to-1 correspondences. Clio displays the union of queries that forms the resulting mapping for corresponding portions in the source schema in separate columns on the right side of target schema tree in its GUI , while SCIA simply uses the match lines with details accessible from clicking the line or the target node. In addition, constraints play extremely important roles in relational schema mapping, and Clio displays constraints explicitly using lines on its GUI. Finally, the cardinality information for each schema node is not as

explicit as its name and data type. This originated from Clio's relational model, since each field occurs only once implicitly in any record. However, Clio supports both relational and XML data, so it is desirable to display cardinality information explicitly.

IV.H SCIA

We now analyze our own system SCIA. We first describe its minimal model, discuss its interface design, and then provide a semiotic analysis, in more detail than for other tools.

IV.H.1 Minimal Model

SCIA stores the full access path to each node, unlike some other tools do not identify nodes with the same name but different path contexts. SCIA does both schema matching and view generation. It has models for both simple 1-to-1 matches and complex n-to-m cases involving functions and conditions. In the automatic matching process, SCIA determines node pairs (<src-elt>, <tgt-elt>) that are related, where <src-elt> is a path in the source schema, and <tgt-elt> is a path in the target schema. Through its context check process, SCIA adds semantic information for data transformation, including joins and grouping conditions and functions by interacting with users at critical points, i.e., turning schema matches into a mapping. In SCIA, a mapping consists of a set of mapping elements of the form described in Section II.A:

([<src-elt>,] [<fcn>] , <fcn>, <src-elt> [, <fcn>]* [, <cdn>], <tgt-elt>)

It allows combining the content of multiple source elements to a single target element or multiple ones. A mapping element can also be represented in the following simpler way, in which there may be only one function for one target element in a mapping element:

```
({<src-elt>,}*  [<fcn>] [, <cdn>], <tgt-elt>)
```

The occurrence of multiple mapping elements for the same target element <tgt-elt> implicitly means the union of all of them. Notice a mapping element for a <tgt-elt> may not necessarily involve a <src-elt>, i.e., it might have a 0-to-1 match, the target value can be a constant or generated by a function. <fcn> covers constants for simplicity.

SCIA uses domain thesaurus. This is optional, but if a good domain thesaurus containing a synonym table, hyponym table and an abbreviation table is provided, it will significantly help with automating the matching process. SCIA has thresholds, coefficients and running modes: interactive or non-interactive. It creates, saves and reuses mappings, and detects and resolves critical points through context check in the mapping process. SCIA also generates and executes query for data transformation.

The above overview suggests that the SCIA source semiotic space consists of the following sorts as shown in Figure IV.8:

1. a top sort, which is the whole system, called SCIA;
2. 5 second level sorts:
   (a) Schemas, which include a source schema and a target schema; each schema consists of schema elements and constraints;
   (b) Mappings, which describe how the source schema and the target schema correspond and how to transform from the source schema to the target schema; each mapping consists of a set of mapping elements;
   (c) Views, which describe how to transform data from the source schema to the target schema using a specific query or programming language;
   (d) Options, which include domain thesaurus, configuration, etc.; and
   (e) Critical points, where user input is necessary and maximally useful; critical points are discoverd as the mapping process proceeds;

3. third level sorts:

   (a) Schema elements, each having a name, data type, cardinality and possibly description;

   (b) Mapping elements, each containing some source element(s), a function, a condition and a target element;

4. and more third level and lower level sorts, which we do not bother to list.

The SCIA semiotic source space also contains constructors, such as

1. a constructor for the top level sort, from schemas, mapping, view, options and critical points to the system;

2. a primary constructor for a second level sort mapping, from a pair of schemas, user input matches, functions and conditions, existing mappings, domain thesaurus and critical points to mapping;

3. a constructor for sort mapping, from a set of mapping elements to mapping;

4. a constructor for a second level sort critical points, detecting and resolving critical points, called context check, which is an important part of the primary mapping constructor;

5. some constructors for a second level sort schema, from relations or elements to schemas, and from fields to relations, which have parameters, e.g., a boolean value indicating source or target;

6. a constructor for a third level sort mapping element, from source schema elements, target schema elements, functions and conditions to mapping elements;

7. some constructors for a fourth level sort schema element, from sub-elements, names, data types, cardinalities, descriptions to elements; the node name has the highest priority among all the properties.

The source semiotic theory also consists of some axioms, such as constraints on the schemas, constraints on each relation, fields and elements, including dependencies, cardinality, and value constraints.

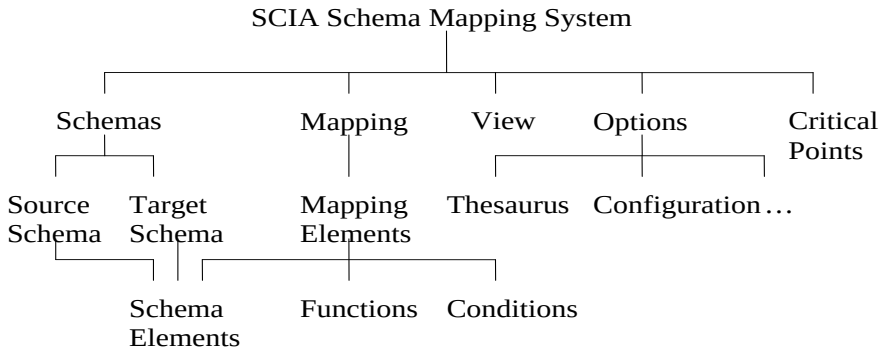

Figure IV.8 SCIA Source Semiotic Space

IV.H.2 Interface Analysis

The main user interface of SCIA is shown in Figure IV.9. The menu bar and submenus are shown in Figure IV.10. The menu bar contains 5 entries: "Schemas", "Mapping", "View", "Options" and "Critical Points". The "Schemas" menu has submenus: "Source" and "Target" for choosing a source schema and a target schema respectively. The first submenu of "Options" is "Domain", which is used to input a domain thesaurus or select one from a list of existing ones. The "Options" menu also contains "Configuration" for adjusting thresholds and coefficients, and it has entries for choosing the running mode of the system: interactive or non-interactive. The "Mapping" menu has submenus "Create", "Open", "Save" and "Clear" for creating, saving mappings, choosing an existing mapping from the mapping library to be reused in the current mapping task, and deleting mappings. Once a file for an existing mapping is selected, it is checked for applicability to this mapping task and is displayed visually in the mapping window. The mapping process will then start from it instead of from scratch. The "View" menu has submenus: "Create", "Check" and "Execute" for creating, checking and executing

views for data transformation.

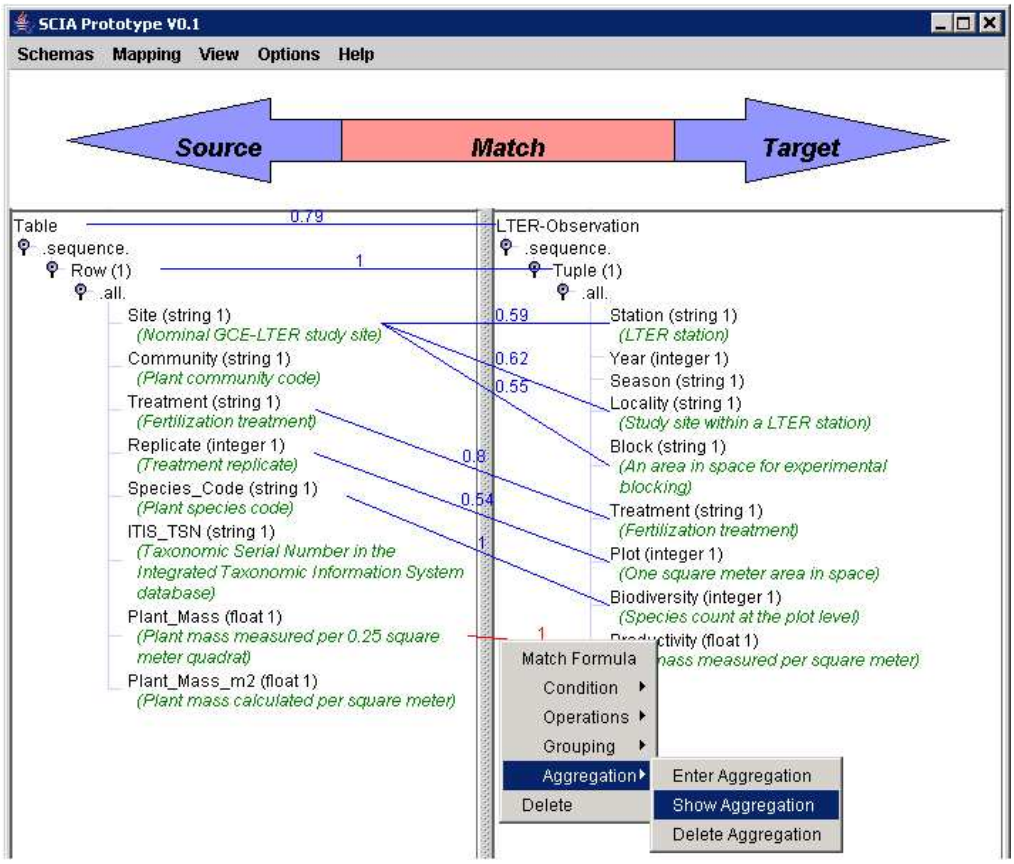

Figure IV.9 SCIA main GUI

Each of the second level sorts in the source semiotic space corresponds to a menu in the menubar, except for Critical Points. Schemas and Mapping occupy the most space in the main GUI, because they deserve special attention, as two main second level sorts, and also as the highest priority arguments of the constructor for the SCIA system.

The two schemas are displayed in the form of trees to visualize their structure as in the COMA++ and Clio GUIs, with the source schema on the left and the target schema on the right in the big white split window in the center of the main GUI. In general, DTDs and XML Schemas are graphs since there are often shared elements and types, but documents are trees. It is often desirable to view the graph structure as a tree structure, by showing multiple versions of the

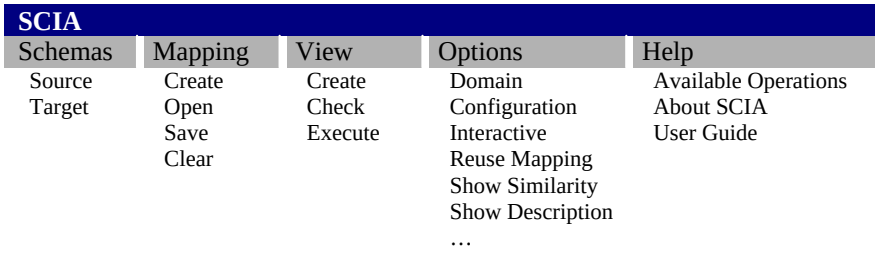

Figure IV.10 SCIA target semiotic space: menus and submenus

shared elements instead of multiple links to single version of a shared element; this is because a tree is more straightforward for human eyes to perceive than a graph, and more important, in SCIA we treat the shared elements as different ones because they have different path contexts that give them different meanings. Indentation, along with the grey dash lines is intended to indicate the parent-child relationship between nodes in a schema tree; this representation is familiar to many users, due to the common file system display in Windows. The icons for nodes can also be changed to the file folder icons used in Windows file system display, a "closed" icon means that the node is collapsed, while an "open" one means that the node is expanded with all its direct children displayed. The most important properties, i.e., data type, cardinality and description of schema elements are displayed near the node names. The priorities of different properties are also reflected. The data type and cardinality are in parenthesis to the right of the node name, while the description text is displayed in different color and different font, in parenthesis and in a seperate line below each node name.

Mappings are represented as blue lines across the middle bar in the split window for schema trees in the SCIA main GUI, with one line for each mapping element. The numeral on each line represents the confidence of this correspondence, with a similarity value from 0, indicating strongly dissimilar, to 1 indicating strongly similar. The primary constructor for the sort Mapping corresponds to the submenu "Create" in the "Mapping" menu and the "Match button" in the middle of the arrow panel right below the menu bar. The two windows for schema trees

are individually scrollable, so that large schemas and mappings between them can be navigated conveniently.

The convention on flow direction in common reading and writing is followed to implicitly indicate that data transformation is from the source on the left to the target on the right. This is consistent with the semantic interpretation of the graphic metaphor for matches at the data level. It may be desirable to add an arrow for each line on the target side, in order to make the direction explicit to users, and avoid potential confusion instead of relying on a convention that does not hold in some other cultures. An alternative is to explicitly show the mapping direction in the arrow panel, by changing the middle part to a dynamic arrow that always displays the current direction, and modifying the source part and target part into some non-arrow objects, such as a square with a tree of dots inside, or simply a square. The richer information supporting data translation, such as functions and conditions in mapping elements, are accessible through a pop-up menu, as in Figure IV.9. At any time, users can input and edit the mapping by clicking the nodes and the lines. When a line is clicked, it will turn from the original blue color into red. With a right click of a mouse, some options for displaying and editing this correspondence and its details, including any functions and conditions will be shown as a pop-up menu.

As discussed earlier, the SCIA minimal model contains an important second level sort for critical points and its constructors. However, they are not accessible from the static main GUI due to their dynamic nature. They appear during the mapping process. Typical interfaces at critical points are shown in Figures III.3, III.4, and III.11. The nodes and match lines highlighted in red identify the nodes on which the context check algorithm is working and for which user feedback is required to adjust the predictions made by the algorithm. Users can respond conveniently by reading the provided information and questions, and then simply choosing an option in a list displayed in the pop-up window based on their understanding of the mapping task. (In an earlier version, we highlighted nodes

in focus by drawing a red arrow on the right side of each concerned node, but according to the general "Look and Feel" principle, the uniform way for highlighting helps users understand the intention of highlighting, so we have changed it to highlight both nodes and lines in the same way.)

The SCIA minimal model for mappings is relatively complicated; a mapping element may involve multiple source elements, multiple functions in a certain order, and conditions for a single target element. It is desirable to display the current focus of action, in order to help users understand the mapping process, and so that they can provide feedback easily. In SCIA, the current focus of action is indicated by the large arrow. When source nodes are clicked by the users or being worked on by the context check algorithm, the source part of the arrow panel is highlighted in pink. When the target nodes are clicked by users or being worked on by the context check algorithm, the target part in the arrow panel is highlighted in pink. When functions and/or conditions are added to obtain semantically correct transformation of data from the source to the target, the middle part of the arrow panel is highlighted. By clicking a part of the arrow panel, users can tell the system they wish to do the corresponding action: selecting nodes from the source, selecting the nodes from the target, or formulating an n-to-m match that involves all the selected n source nodes and m target nodes.

The primary view constructor corresponds to "Create" in the "View" menu; when clicked, the mapping generates an executable view of the target over the source, in the XQuery format, and then displays the generated view in an editable window, the View Editor, for users to check and edit if needed; see Figure III.6. The changes made by users in the View Editor can be saved by clicking the "Save" button or cleared by clicking the "Reset" button; clicking the "Cancel" button will discard any changes and close the View Editor. "Execute" in the "View" menu launches an XQuery engine to execute the generated view and transform a source document into an instance of the target schema, and then display it in the Query Result Viewer. Currently, Quilt, an ancestor of XQuery, and its query

engine Kweelt are used in SCIA. The query result can also be edited and saved by clicking the Save button in Query Result Viewer, shown in Figure III.7.

IV.I Schema Mapping Tool Interface Design Principles

According to the analyses of the SCIA, Rondo, COMA++ and Clio interfaces in the above sections, these different tools are based on slightly different minimal models for schemas and very different minimal models for mappings, and they perform different functionalities. For example, Rondo and COMA++ do just schema matching, Clio does just view generation, while SCIA does both schema matching and view generation. However their source semiotic spaces and their structures have very much in common. For example, schemas and mappings are two common main second level sorts, they are also common main arguments for constructing these entire systems. A schema contains a set of schema elements in hierarchy, and a schema element has properties of name, data type, parent, children, cardinality and possibly description. Hence, according to the Principle Sort Preservation in algebraic semiotics, schemas and mappings are the most important to visualize in the target space, and the parent-child relationships between schema elements should be displayed. All the above systems have primary constructors for mapping and/or view. According to Principle Constructor Preservation, these constructors should be emphasized in the target space. Schemas may have constraints, and constraints play important roles in finding mappings between schemas. According to Principle Axiom Preservation, these constraints should be displayed, like Clio does. In order to understand how elements of two schemas relate at the level of data translation, users need to acquire more knowledge about the data than could be seen from the schemas alone. As discussed in Section III.G.7 about source data, showing instances in the GUI was thought a good thing to do in SCIA. However source data is not of a high level sort in the source space, its priority is below that of schemas. Schemas are form, data are content. According to Principle

F/C and Principle HL/LL, if showing data instances would clutter the display of schemas or the interface, it has to be sacrificed, by providing other means for users to access them. The following four principles summarize some most important considerations specific to the design process for a schema mapping tool interface.

1. Mapping preservation: the most important aspects of mappings should be preserved. More specifically, the mapping structure should be preserved first, and also the structure of each mapping element, i.e., its source elements and target element. The details of each mapping element, i.e., how source elements are combined to a target element under what condition if supported, should also be preserved, but secondly. For example, in SCIA, the details of a mapping element, such as functions and conditions, are accessible via clicking the match line, though not displayed in the GUI directly as Clio does. Since Clio allows only Skolem functions for handling missing values, functions and paths involved are not complicated, it is nice to show the details on GUI directly. In SCIA, the formula for a local n-to-1 match can be too long to be displayed neatly in a narrow column on GUI.

2. Schema preservation: the hierarchical structure of schemas should be preserved.

3. Schema element properties preservation: the most important properties of schema elements should be preserved. For example, node names, data types, cardinalities and descriptions should be preserved.

4. Schema constraints preservation: the most important constraints should be preserved. For example, primary key and referential constraints should be preserved.

IV.J Summary

In this chapter we introduce algebraic semiotic approaches to interface design and analysis, and analyze the interfaces of several current schema mapping tools, including Rondo, Clio, COMA++, and our own SCIA, using notions of the minimal model needed to support a tool, and of a semiotic morphism from this model into the interface design space. On this basis, we discuss design principles for schema mapping user interfaces.

Chapter V

Related Work

This chapter discusses related work on schema mapping algorithms, systems and semantics. First we review schema matching algorithms and then existing prototypes, and exsiting algorithms and tools that address query discovery. Since the semantics for schema mapping is important for system development, we also summarize the work on semantics for schema mapping. Finally we compare our approach with existing ones.

V.A Schema Mapping Algorithms and Systems

Most mapping algorithms just find interschema element correspondences, i.e., they are matching algorithms. At most, matching algorithms represent a mapping as a similarity relation over the powersets of the nodes of input schemas $\mathcal{S}$ and $\mathcal{S}'$; actually, most of them focus on finding only the most common case, i.e., 1-to-1 matches, in a format like (e, e', v) where e is an element from $\mathcal{S}$, e' is an element from $\mathcal{S}'$, v is a similarity value usually in the range of [0, 1], with a higher v indicating that e and e' are more similar. There are many different matching algorithms, using single criterion (individual matchers) or using combination of individual matchers. An extensive review of techniques and tools for automatic schema matching up to 2001 is given in [82]. Detailed classfication criteria for individual matchers are also given in [82]. Here we classify individual matchers as

follows.

1. **Instance *vs* schema**: matching algorithms can consider instance or schema information. Schema-level information includes schema structure and the properties of schema elements, such as name, description, data type, and constraints. Instance-level information includes the statistics of data content, e.g., maximum, minimum, average and variance for numeric elements, word frequencies and key terms for string elements.

2. **Rule-based *vs* learner-based**: Matching algorithms can employ handcrafted rules or machine leanrning techniques. [18] classifies matching algorithms into rule- or learner-based. Rule-based approaches usually use rules that exploit schema information. A sample rule is "two elements match if their names are the same" in a *Name* matcher. Learner-based approaches may exploit information in both schema and data. An individual learner-based matcher usually employs a single learning technique (e.g., neural networks or naive Bayes) to match schema elements based on schema-level information and/or instance-level information [19, 18, 55, 56, 7, 8, 16]. An unusual learner-based matcher uses statistical methods to find the underlying unified schema model from many schemas in a sepecifc domain; the assumption is that there exists a schema model that describes how schemas are generated from a finite *vocabulary* of elements in a specific domain, and hence gives correspondences of elements across all schemas in that domain [48].

3. **Element *vs* structure**: Matching can be performed for individual schema elements using linguistic matching techniques, or for combinations of elements, such as schema structure using structural matching techniques; examples are tree matching [88] and graph matching [62].

Many schema matching systems use a combination of multiple individual matchers to exploit multiple matching criteria, and perform matching at both

element level and structure level. In addition, most schema matching systems take as input also auxiliary information, such as dictionaries, ontologies, previous match results, and user input. The main differences among them are a) rule- or learner-based, and b) using instances or not.

Rule-based systems usually operate on only schema information, and so run fairly fast. But it is hard for them to exploit the rich information in data even when it is available. The DIKE system [75, 76] computes the similarity between two schema elements based on the similarity of the characteristics of the elements and the similarity of related elements. ARTEMIS in MOMIS [6] computes the similarity of schema elements as a weighted sum of the similarities of name, data type, and substructure. The Cupid system [59] combines linguistic and structural matching techniques, employs criteria based on names, data types, and hierarchical structure of schemas, and also uses domain thesauri. One of the main operations for model management, *Match*, in Rondo [63] uses a graph matching algorithm, *Similarity Flooding* [62], and also uses name matching to provide initial matches. The Coma system [17] provides a flexible architecture for combining multiple individual matchers.

Learner-based matching systems usually exploit instance-level information. The DELTA system [16] is an exception, it associates with each schema element a text string that consists of the element name and all other meta-data on the element, and then matches elements based on the similarity of the text strings. The SemInt system [56, 55] uses a neural-network learning approach, exploiting both schema- and instance-level information. The Autoplex and Automatch systems [8] use a Naive Bayes learning approach that exploits data instances to match elements. The LSD system for 1-to-1 matching [19] combines complementary rule-based approaches and learner-based approaches; and its multistrategy framework, subsequently extended in COMAP for complex matching and GLUE for taxonomy matching [18], employs multiple base learners to make matching predictions, then combines their predictions using a meta-learner. LSD, COMAP and GLUE

requires user supplied mappings from some data sources to the gloabal schema, from which a learner is trained by discovering characteric instance patterns and matching rules, and then used to match other data sources to the global schema. Since learner-based approaches usually rely on not only schema information, but also both source and target data, they are not applicable to data transformation and integration, because target data is not available before data transformation or integration is done.

Each individual matching technique has its own strength and weakness, hence more and more matching tools are being assembled from multiple components [17, 20, 86]. Recently more effort is on the engineering issues of schema matching, how to combine multiple components, such as COMA++ [4] and Protoplasm [10], and how to tune a schema matching system for a domain, like eTuner [86].

There are many fewer algorithms and systems for schema mapping at the level of query discovery (or view generation, or transformation code generation). Clio [46, 64, 100, 81, 80, 47] is a representative system that generates SQL or XQuery expressions that can be excecuted to translate data from one schema to another. Given schema matches obtained from a knowledge base or entered by user through a GUI, Clio selects enough of the matches to cover a maximal set of columns or elements of the target schema, suggests join clauses to tie together components of the source schem and computes all possible mappings using traditional DBMS optimization techniques based on the semantics of schemas including various constraints, and allows users to select a mapping conforming to domain specific semantics and to verify correctness of the generated view for the target over the source by checking example results. MapForce [3] and Stylus Studio [90] in industry are similar to Clio; in addition, they provide visual support for formulating functions. Incorporating domain-specific characterics of XML documents, such as domain ontology, common transformation types, and specific DTD constructs, Xtra discovers a sequence of operations for transforming a source DTD into

a target DTD via tree matching, and then uses the operation sequence to generate an equivalent XSLT transformation script [91]. We have not found experimental results from Xtra supporting its claim that user interactions are avoided, making it a big question how well it works.

V.B Semantics for Schema Mapping

Compared with the enormous effort in designing algorithms and developing systems for schema mapping, there is much less work on studying semantics for schema mapping. In [82] Rahm and Bernstein defined a schema mapping to be a set of *mapping elements*, each of which indicates that certain elements of schema $\mathcal{S}$ are mapped to certain elements of another schema $\mathcal{S}'$. A mapping element can have a *mapping expression* which specifies how the $\mathcal{S}$ and $\mathcal{S}'$ elements are related. The mapping expression may be a directional, i.e., function, or non-directional, i.e., relation, may use whatever relationships (e.g., $=$, $\subseteq$, *isa*, *partof*) and functions (e.g., addition, concatenation) that are defined in the expression language being used. Madhavan et al in [58] defined a schema mapping to consist a set of relationships between expressions over two schemas, in the form of x **op** x' where x and x' are expressions over elements of $\mathcal{S}$ and $\mathcal{S}'$ respectively, which are actually equivalent to the non-directional mapping expressions in [82]. The operator **op** is well defined with respect to the output types of x and x'. For example, if both expressions have relations as output types, then **op** can be $=$ or $\subseteq$. If x outputs a constant and x' outputs a unary relation, then **op** can be $\in$.

Helper representation was introduced in [58] and used in [19, 18]; this is a user domain representation that provides well-defined semantics for the notion of similarity and mappings that involve operators from different representations. Authors in [19, 18] defined a similarity measure *Sim* over the concepts in a given user domain representation $\mathcal{U}$, under the assumption that for any schema matching solution, the user can map any two concepts e and e' of the input schemas into

semantically equivalent concepts $M(e)$ and $M(e')$ in $\mathcal{U}$. They define the *matching problem* as: for each element e of $\mathcal{S}$, find e' that maximizes $Sim(M(e), M(e'))$ among all elements of $\mathcal{S}'$, in other words, find e' such that the similarity value computed by Sim between the equivalent concepts in $\mathcal{U}$ is the highest. Thus, when a solution produces a semantic mapping (e, e', v), v can be interpreted to be the best estimation of the true similarity value $S(M(e), M(e'))$ that the solution could produce.

Alagic and Bernstein [2] study schemas and schema morphisms using institutions [36]. They define a schema as a pair of schema signature and a set of integrity constraints. A schema signature specifies structural and operational features of a schema, with signatures for data types and their operations and signatures for database sets (relations, collections, etc.) as typical components. Integrity constraints are expressed as sentences, with forms determined by the chosen logic and syntax of the constraint language. Using category theory, they define schema morphisms that map the structural properties and integrity constraints of schemas and database morphisms that map the actual data. This is the closest to our semantics for schema mapping [40, 34] developed by Goguen as part of the SCIA project. our *abstract schemas* axiomatize the notion of a kind of schema (e.g., relational, XML, spread sheets, etc.). Therefore, they naturally serve as user domain representations. They include all schema kinds in common use (e.g., relational schema, XML DTD and Schema, ontologies). All kinds of schema are treated in a uniform way once they are translated into abstract schemas. More important, *abstract schema morphisms* cover morphisms between different kinds of schemas, i.e., *heteromorphisms*. Our schema morphisms cover both schema morphisms and database morphisms in [2], and specify both interschema correspondences (i.e., objectives of schema mapping at the concept level), and data transformation steps (i.e., objective of schema mapping at data level). These greatly extend the closest work in [2], in which the authors do not give single general definition for schema or morphism, but requires a separate definition for each kind of schema, and mappings

between different kinds of data model require handling by institution morphisms rather than schema morphisms. The logic-based schema mappings of [58] are relations that cause very high complexity for query discovery, whereas our schema morphisms are functions, which makes them easy to use (e.g., by translation into XQuery).

Popa and Fagin et al interpret interschema correspondences as source to target dependencies in Clio project [81, 23, 24]. Given a source instance, there may be many solutions to the data exchange problem, that is, many target instances that satisfy the source-to-target dependencies and the target constraints. They give an algebraic specification that selects, among all solutions to the data exchange problem, a special class of solutions that they call universal. They show that a universal solution has no more and no less data than required for data exchange and that it represents the entire space of possible solutions [23]. All universal solutions have the same core (up to isomorphism); they show that this core is also a universal solution, and hence the smallest universal solution. They identify natural and fairly broad conditions under which there are polynomial-time algorithms for computing the core of a universal solution [24]. Gottlob and Nash use the technique of hypertree decompositions to derive improved algorithms for computing the cores [42, 43]. Fagin et al study composition [25] and inverse of source-to-target dependencies in [22]. Nash et al [68] extend [25] and study composition of mappings given by embedded dependencies, which are tuple generating dependencies that may have new symbols in the conclusion. Notice that none of these studies touch mappings that involve complex matches with functions as our SCIA project does.

V.C Comparison with Prior Work

Our approach to schema mapping in SCIA (Chapter III) extends and combines techniques from the above prior work, also makes several contributions.

1. We introduce the idea of *critical points* for schema mapping. Most previous studies have focused on developing automatic matching algorithms. They either ignore the issue of user interaction, or treat it as an afterthought, or at most use user feedback as a separate matcher. However, it has been widely recognized that schema mapping, including both schema matching and query discovery, entails making decisions that require user input. Several recent works in ontology matching [61, 72, 73, 74] and query discovery [46, 64, 100, 81, 80] treat user feedback as an integral part of the matching or query discovery process. For instance, PROMPT [72] frequently solicits user feedback on its matching decisions (e.g., confirm or reject the decisions), then makes subsequent decisions based on the feedback; Clio relies on users to input matches, to select from generated possible mapping queries, and to verify the correctness of mapping queries. In fact, efficient user interactions are crucial to the success of the whole mapping process. Hence our key innovation regarding the issue of user interaction is that we develop techniques to detect when and where user interactions are maximally helpful, i.e., critical points, and to provide users with plenty of information to guide interaction at these points, thereby minimizing the total amount of manual effort required. The detection of critical points is done iteratively, based on current combined results computed from multiple algorithms and user feedback, until the user is satisfied that a correct mapping has been created.

2. We bridge the gap between schema matching and view generation by handling both in a single tool. As discussed above, most schema mapping algorithms and systems are actually for schema matching, to find interschema correspondences; only a few systems generate mapping queries for translating data from one schema to another, but they don't handle schema matching. Instead, they require interschema correspondences as input. SCIA outputs a mapping as an interschema correspondence as well as an executable view in XQuery format for immediate data transformation.

3. We handle complex mappings without requiring domain knowledge as initial input. The vast majority of prior work focused only on finding simple 1-1 matches. Several studies [65, 99, 11] deal with complex matching in the sense that matches are hard-coded into rules or user-provided domain ontologies. COMAP [18] finds complex matches based on user input of the relationships between attributes and real data (including the target instance that is not available as input in the context of data transformation and virtual data integration). COMAP checks which elements' data are consistent with user input relationships, and then it outputs the complex matches for those elements. Clio [46, 64, 100, 81, 80] assumes that matches have been given, and focuses on finding the mapping queries. In SCIA, domain knowledge about complex matches is an optional initial input, which can greatly help with the mapping process when available. Our contribution is in techniques to identify where complex matches most likely exist, and then asking users specific questions, and assisting users to formulate them.

4. We explicitly exploit the context information embedded in paths, and we combine multiple algorithms to exploit multiple aspects of schema information in a novel way through our context checker.

5. We have semantic models for schemas and mappings in algebra [34, 40], which builds on previous work, but enhances it. See comparisons in the above section.

Chapter VI

Conclusion

VI.A Main Contributions

The major contribution of this dissertation is an approach that bridges the gap between the manual schema mapping used in practice due to the shortage of satisfying solutions, and recent automatic mapping techniques that ignore complex mappings or require expensive or unavailable information for finding complex mappings. It advocates *critical points* for schema mapping where user input is necessary and maximally useful, in order to save total user effort through helping users at such points. It also provides a novel way to combine multiple techniques, each exploiting a certain type of information, through context check to find critical points and create mappings. Up to now, it is the only approach that supports both interschema correspondence discovery (i.e., schema matching) and query discovery (i.e., view generation), which is greatly desirable for data transformation and integration. It can handle XML DTD and Schema, relational schema and ontologies in the matching step; it can handle XML in the view generation step now. It has potential for handling any kind of schema because of its underlying semantics [34, 40].

In addition, this dissertation contributes to the area of user interface design, by introducing schema mapping as an application of algebraic semiotics. It

constructs the minimal models for familiar schema mapping tools, and their source semiotic spaces; analyzes the semiotic morphisms from those source spaces to the user interfaces of the tools; provides suggestions for improvement; and proposes design principles for schema mapping user interface. This application also gives a model for obtaining user interface design principles for other application areas.

VI.B Remaining Issues

Several issues remain to be solved. First, SCIA currently supports matching between any combination of common representations, e.g., XML DTD or Schema, relational schema and ontology, but the view generation part works only for XML, though the semantic models behind it support mapping between any kinds of schema. We plan to extend it to support at least mapping relational to XML. Second, we will explore the use of more constraints in the mapping process, and provide support for users to enter functions and conditions to matches in a more intuitive way. Furthermore, as more and more data sources have semantics specified in associated ontologies, it seems promising to use ontology matching to help with schema mapping. In addition, since our view generation algorithm works only for Quilt, extending it to standard XQuery would make it more useful in practical applications.

VI.C Future Directions

The next generation of data integration scenarios is targeting communities that need a single point of access to data owned by diverse community members, without incurring the heavy development and evolution cost of existing integration systems. Each member is responsible of registering her independent data source to a community-shared database, without the need of an administrator/developer who manages the integration process. Schema mapping is the key technology to assist data source owners in identifying common entities and rela-

tionships between their source and the community-wide database. Preliminary results in collaboration with the RIDE project at UCSD show that SCIA saves significant user effort in source registration by reducing the number of possible mappings that the user has to choose from. We are planning to addresses the challenges presented by such dynamic environments, where schemas evolve frequently. Incremental update through effective mapping reuse and composition is very important in minimizing data owner's effort. SCIA supports straightforward reuse of mappings, but more functionality should be offered, to save the data source owner's effort in updating the mapping by providing help deciding whether it is better to reuse the old mapping, or to use a mapping associated with different but similar schema. While more and more effort is being made to create community-wide ontologies, it is very promising to use them to further save user effort in the creation of semantic mappings between schemas. SCIA now utilizes domain thesauri and supports ontology matching, We plan to extend it to use formal ontology directly in the schema mapping process, and to use the ontology matching results to provide semantic clues for the schema mapping process.

We will continue exploring the use of schema mapping techniques for suggesting connections and transformation steps between components in workflow management. Mapping reuse and composition, and using ontologies are extremely desirable in this context too. We are also interested in extending the schema mapping techniques for process/system integration in general. In addition, we will extend the mapping technology to the integration of not only structured and semi-structured data, but also unstructured data. Abstract Schemas and Schema Morphisms [34, 40] born in SCIA project cover different kinds of schemas, such as relational XML, spread sheets and flat files, and even ontologies. We plan to develop a new schema specification language and fully implement abstract schemas and schema morphisms in SCIA. Efficient user interaction is an integral part in reducing the total user effort in schema mapping, and we will continue research on user interface design.

Appendix A

Schemas Used in the Formal SCIA Experiment

book1.dtd

```
<!ELEMENT bib  (book*)>
<!ELEMENT book  (title, publisher, (author+ | editor+ ), price)>
<!ATTLIST book  year CDATA  #REQUIRED
                id CDATA    #REQUIRED>
<!ELEMENT author  (last, first )>
<!ELEMENT editor  (last, first, affiliation )>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT last  (#PCDATA )>
<!ELEMENT first  (#PCDATA )>
<!ELEMENT affiliation  (#PCDATA )>
<!ELEMENT publisher  (#PCDATA )>
<!ELEMENT price  (#PCDATA )>
```

book2.dtd

```
<!ELEMENT arts (book | article) >
<!ELEMENT book (author+, title, publisher, price)>
<!ATTLIST book year CDATA #REQUIRED>
<!ELEMENT article (author+, title, year?,
                   (shortversion|longversion))>
<!ATTLIST article type CDATA #REQUIRED>
<!ELEMENT publisher (name, address)>
<!ELEMENT author (firstname?, lastname)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT shortversion (#PCDATA)>
<!ELEMENT longversion (#PCDATA)>
```

book3.dtd

```
<!ELEMENT bookstore (book*)>
<!ELEMENT book (title,author*, editor*, publisher,price)>
<!ATTLIST book genre CDATA #IMPLIED
               id CDATA #REQUIRED
               year CDATA #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (name)>
<!ELEMENT editor (name)>
```

```
<!ELEMENT publisher (name, address)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```

book4.dtd

```
<!ELEMENT bookstore (book*)>
<!ELEMENT book (title,author*, editor*, publisher,price)>
<!ATTLIST book genre CDATA #IMPLIED
               year CDATA #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (name)>
<!ELEMENT editor (name)>
<!ELEMENT publisher (name, address)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```

book5.dtd

```
<!ELEMENT arts (book | article) >
<!ELEMENT book (author+, title, publisher, price)>
<!ATTLIST book year CDATA #REQUIRED
               id   CDATA #REQUIRED>
<!ELEMENT article (author+, title, year?,
                   (shortversion|longversion))>
<!ATTLIST article type CDATA #REQUIRED>
<!ELEMENT publisher (name, address)>
<!ELEMENT author (firstname?, lastname)>
```

```
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT shortversion (#PCDATA)>
<!ELEMENT longversion (#PCDATA)>
```

books_book.dtd

```
<!ELEMENT arts  (book*)>
<!ELEMENT book  (title, writer+, publisher, price, description,
                 comments)>
<!ATTLIST book  year CDATA  #REQUIRED>
<!ELEMENT writer  (lastname, firstname, nationality)>
<!ELEMENT title  (#PCDATA)>
<!ELEMENT firstname  (#PCDATA)>
<!ELEMENT lastname  (#PCDATA)>
<!ELEMENT nationality  (#PCDATA)>
<!ELEMENT publisher  (name, address)>
<!ELEMENT name  (#PCDATA)>
<!ELEMENT address  (#PCDATA)>
<!ELEMENT description  (#PCDATA)>
<!ELEMENT comments  (#PCDATA)>
<!ELEMENT price  (#PCDATA)>
```

books_book3.dtd

```
<!ELEMENT bookstore (book*)>
<!ELEMENT book (title,writer*, editor*, publisher,price)>
<!ATTLIST book  genre CDATA #IMPLIED
                id CDATA #REQUIRED
                year CDATA #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT writer (name)>
<!ELEMENT editor (name)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```

books_author.dtd

```
<!ELEMENT store  (author*)>
<!ELEMENT author  (book+, name)>
<!ELEMENT title  (#PCDATA)>
<!ELEMENT name  (#PCDATA)>
<!ELEMENT book (genre, title, publisher, price, abstract?,
                review?)>
<!ATTLIST book  year CDATA  #REQUIRED
                id   CDATA  #REQUIRED>
<!ELEMENT publisher  (#PCDATA)>
<!ELEMENT abstract  (#PCDATA)>
<!ELEMENT review  (#PCDATA)>
<!ELEMENT price  (#PCDATA)>
<!ELEMENT genre  (#PCDATA)>
```

books_author2.dtd

```
<!ELEMENT store  (author*)>
<!ELEMENT author  (book+, firstname, lastname)>
<!ELEMENT title  (#PCDATA)>
<!ELEMENT firstname  (#PCDATA)>
<!ELEMENT lastname  (#PCDATA)>
<!ELEMENT book (genre, title, publisher, price, abstract?,
                review?)>
<!ATTLIST book year CDATA  #REQUIRED
               id   CDATA  #REQUIRED>
<!ELEMENT publisher  (#PCDATA)>
<!ELEMENT abstract  (#PCDATA)>
<!ELEMENT review  (#PCDATA)>
<!ELEMENT price  (#PCDATA)>
<!ELEMENT genre  (#PCDATA)>
```

Appendix B

Layout of the Formal SCIA Experiment

Layout

The overall layout of the experiment is as follows in chronological order. Total time is expected to be about 65 minutes.

1. The subject is greeted. (5 min)
2. The facilitator guides the subject through doing two common sense schema mapping tasks (demo tasks A and B) in two modes. These two demo tasks cover all features involved in the following test tasks (GUI test tasks A and B, timing tasks A and B). (20 min)
3. The subject starts the first part of the experiment - evaluating the user interface. If the subject has any questions or comments during conducting of the experiment, they are recorded. Whenever the subject does not know how to move forward, the facilitator asks what is the problem, and gives direction in order to proceed. (20 min)
    - The subject runs GUI test task A using critical points;

- The subject runs GUI test task B using critical points.

4. The subject starts working on the second part of the experiment - perform timing task with critical points if the subject belongs to group II, otherwise without using critical points. This time the subject is familiar with the GUI and how to manually input matches, conditions, functions, etc. The time the subject takes to complete each task correctly is recorded. Whenever the subject does not know how to move forward, the facilitator gives direction in order to proceed as fast and smoothly as possible. (15 min)

5. If the subject finishes the above sessions within 50 minutes, we ask them to run the same timing tasks again in different way. If the subject runs the timing tasks with critical points first, now runs the same tasks without critical points; if the subject runs the timing tasks without using critical points, now runs them with critical points. This session is added when we find that some subjects finish the experiments within 40 to 50 minutes, some of them wonder how the different way works for the same task, and we also want to know whether with or without critical points and order any difference for the time taken for the same task for the same subject.

6. The subject is given a questionnaire to record his/hers overall opinion of the graphical user interface of SCIA and to record any general comments he/she might have. (5 min)

Roles and Responsibilities

Greeter The role of the greeter is to meet the subject, explain the purpose and the general format of the test, and conduct the pre-test questionnaire. The goal of the greeter is to empower the subject, relieving any anxiety he/she might have. The greeter should read the purpose and the format of the test from a written script so as not to forget any important information. The

following are the contents of this script:

Thank you very much for your interest and help! Our tool SCIA is for assisting users create semantic mapping between a pair of schemas, the output mapping can be executed to transform data from one schema to another. SCIA detects critical points where user input is necessary and maximally useful and help users at those hard points through a GUI, while automating the easy points as much possible. We ask you to help us evaluate the user interface and how much user effort is saved by using critical points.

We will first show you how SCIA works by running 2 representative tasks from beginning until the correct mappings are generated. Then you will try to run 2 slightly different tasks using critical points for evaluating the GUI. After this, you will be familiar with the tool, you will run the last two tasks using critical points (or without using critical points), and we will record the time that you take to finish it.

Please remember if you met any difficulty on moving forward, it is not your fault, it means there are problems with the design of the tool. Please let us know any trouble you met, that will help us improve the tool.

After you conduct the test, you will be asked to rank the effectiveness of the graphical user interface using a questionnaire. If you have any comments or questions before, during, or after the test, please don't hesitate to ask us. You comments will be recorded for evaluation purposes.

Facilitator The facilitator is the person who actually conducts the test and the only person who speaks to the subject during the test. The facilitator demonstrates the tool and presents the test cases to the user one at a time.

The facilitator encourages the user to "think aloud". However the facilitator should not answer the subject's questions about how to use the interface, or guide the user in the first part for evaluating the interface. If the user gets "stuck" during executing a procedure, the facilitator should ask the user

what's causing his/hers inability to continue, and then allow the user some time to figure out how to proceed. If the user is still unable to perform a step however, the facilitator should execute the step for the user so that the experiment can move on. In the second part, whenever the subject meets any trouble, the facilitator provides help in order to move forward as fast and smoothly as possible. The facilitator should maintain neutral demeanor, not showing to the user if the actions he/she performs are correct or incorrect.

Observer The role of the observer is to take careful notes and to video tape the experiment with focus on the subject and the screen. The notes should be taken on index cards, with one observation per card, so that they could be sorted after the test. The observer should record any user comments or thoughts, and the use case step at which the comments were made. If the subject encounters a difficulty, the observer records the step, the stated reason for the difficulty, and any other relevant information that might help explain the difficulty. In addition, the observer records the time it takes to complete each task.

Appendix C

SCIA Mapping Structure Specification: mapping.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="mapping" type="MappingType">
  </xs:element>
  <xs:complexType name="MappingType">
    <xs:sequence>
      <xs:element name="targetSchema" type="xs:string"
                  use ="required"/>
      <xs:element name="sourceSchema" type="xs:string"
                  use ="required"/>
      <xs:element name="component" type="ComponentType"
                  minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ComponentType">
    <xs:sequence>
```

```
    <xs:element ref="targetPath"/>
    <xs:choice>
      <xs:element ref="constant"/>
      <xs:element ref="functionOutput"/>
      <xs:element name="one-to-one" type="one-to-oneType"/>
      <xs:element name="local-n-to-1" type="Local-n-to-1Type"/>
      <xs:element name="global-n-to-1"
                  type="Global-n-to-1Type"/>
    </xs:choice>
    <xs:element name="secondMatch" type="Basic-1-to-1Type"
                minOccurs="0" maxOccurs="1"/>
    <xs:element name="kthMatch" type="Basic-1-to-1Type"
                minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Basic-1-to-1Type">
  <xs:sequence>
    <xs:element ref="sourcePath"/>
    <xs:element name="similarity" type="xs:float"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="one-to-oneType">
  <xs:sequence>
    <xs:element ref="sourcePath"/>
    <xs:element name="similarity" type="xs:float"/>
    <xs:element ref="operations" minOccurs="0"/>
    <xs:element name="condition" type="ConditionType"
                minOccurs="0"/>
    <xs:choice>
```

```
      <xs:element name="groupAttrs" minOccurs="0">
<xs:complexType>
          <xs:sequence>
            <xs:element name="groupAttr" type="xs:string"
                        minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
      </xs:element>
      <xs:element name="aggregator" type="xs:string"
                  minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
<xs:element name="sourcePath" type="xs:string"/>
<xs:element name="targetPath" type="xs:string"/>
<xs:element name="constant" type="xs:string"/>
<xs:element name="functionOutput" type="xs:string"/>
<xs:element name="operations" type="xs:string"/>
<xs:complexType name="Local-n-to-1Type">
  <xs:sequence>
    <xs:element name="unit1" type="Unit1Type"
                minOccurs="1" maxOccurs="unbounded"/>
    <xs:element name="unitN" type="LastUnitType"/>
    <xs:element name="similarity" type="xs:float"/>
    <xs:element name="condition" type="ConditionType"
                minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Unit1Type">
```

```
  <xs:sequence>
    <xs:element ref="sourcePath" minOccurs="1"/>
    <xs:element name="operations" type="xs:string"
                minOccurs="0"/>
    <xs:element name="combOperation" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="LastUnitType">
  <xs:sequence>
    <xs:element ref="sourcePath" minOccurs="1"/>
    <xs:element name="operations" type="xs:string"
                minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Global-n-to-1Type">
  <xs:sequence>
    <xs:element name="one-to-one" type="one-to-oneType"
                minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="local-n-to-1" type="Local-n-to-1Type"
                minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ConditionType">
  <xs:sequence>
    <xs:choice>
     <xs:element ref="sourcePath"/>
      <xs:element ref="targetPath"/>
    </xs:choice>
    <xs:element name="compareSymbol" type="xs:string"/>
```

```
      <xs:choice>
        <xs:choice>
          <xs:element ref="sourcePath"/>
          <xs:element ref="targetPath"/>
        </xs:choice>
        <xs:element ref="constant"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Bibliography

[1] The US long term ecological research network. `http://www.lternet.edu`.

[2] Suad Alagic and Philip Bernstein. A model theory for generic model management. In Giorgio Ghelli and Gösta Grahne, editors, *Proc. Database Programming Languages 2001*, pages 228–246. Springer, 2002.

[3] Altova. Mapforce. `http://www.altova.com/products-mapforce.html`.

[4] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with COMA++. In *Proc. SIGMOD*, 2005. (Software Demonstration).

[5] C. Batini, M. Lenzerini, and SB. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Survey*, 18(4):323–364, 1986.

[6] Sonia Bergamaschi, Silvana Castano, Maurizio Vincini, and Domenico Beneventano. Semantic integration of heterogeneous information sources. *Data and Knowledge Engineering*, 36(3):215–249, 2001.

[7] Jacob Berlin and Amihai Motro. Autoplex: Automated discovery of content for virtual databases. In *Proc. the Conf. on Cooperative Information Systems (CoopIS)*, 2001.

[8] Jacob Berlin and Amihai Motro. Database schema matching using machine learning with feature selection. In *Proc. the Conf. on Advanced Information Systems Engineering (CAiSE)*, 2002.

[9] Philip Bernstein. Applying model management to classical meta data problems. In *Proc. Conf. on Innovative Database Research*, pages 209–220, 2003.

[10] Philip A. Bernstein, Sergey Melnik, Michalis Petropoulos, and Christoph Quix. Industrial-strength schema matching. *SIGMOD Rec.*, 33(4):38–43, 2004.

[11] Joachim Biskup and David Embley. Extracting information from heterogeneous information sources using ontologically specified target view. *Information Systems*, 28:169–212, 2003.

[12] Shawn Bowers and Bertram Ludäscher. An ontology-driven framework for data transformation in scientific workflows. In *Data Integration in the Life Sciences*. Springer, 2004.

[13] Daniel Chandler. *Semiotics: The Basics*. Routledge, 2001.

[14] Guija Choe, Young-Kwang Nam, Joseph Goguen, and Guilian Wang. Design and implementation of a portal query processing algorithm for distributed semistructured data. to be submitted.

[15] Guija Choe, Young-Kwang Nam, Joseph Goguen, and Guilian Wang. Information retrieval from distributed semistructured documents using metadata interface. In *Proc. of PAKDD*, 2005.

[16] Chris Clifton, E.Housman, and Arnon Rosenthal. Experience with a combined approach to attribute-matching across heterogeneous databases. In *Proc. IFIP Working Conf. on Data Semantics*, 1997.

[17] Hong-Hai Do and Erhard Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proc. 28th VLDB Conference*, 2002.

[18] An-Hai Doan. *Thesis: Learning to Translate between Structured Representations of Data*. PhD thesis, University of Washington, 2003. `http://anhai.cs.uiuc.edu/home/thesis/anhaithesis.pdf`.

[19] An-Hai Doan, Pedro Domingos, and Alon Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proc. ACM Intl. Conf. on Management of Data (SIGMOD)*, 2001.

[20] AnHai Doan, Natalya F. Noy, and Alon Y. Halevy. Introduction to the special issue on semantic integration. *SIGMOD Rec.*, 33(4):11–13, 2004.

[21] Tony Fountain et al. The spatial data workbench: A system for managing and mining large geospatial collections for ecology. In *Supercomputing Conference: High Performance Networking and Computing*, 2002.

[22] Ron Fagin. Inverting schema mappings. In *Proc. the ACM Symposium on Principles of Database Systems (PODS)*, 2006. to appear.

[23] Ron Fagin, Phokion Kolaitis, Renee J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Proc. International Conference on Database Theory (ICDT)*, pages 207–224, 2003.

[24] Ron Fagin, Phokion Kolaitis, and Lucian Popa. Data exchange: Getting to the core. In *Proc. the ACM Symposium on Principles of Database Systems (PODS)*, 2003.

[25] Ron Fagin, Phokion G. Kolaitis Lucian Popa, and Wang-Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, pages 83–94, 2004.

[26] Partnership for Biodiversity Informatics (PBI). Science environment for ecological knowledge (seek). `http://seek.ecoinformatics.org`.

[27] The Apache Software Foundation. Apache lucene. `http://lucene.apache.org/java/docs/index.html`.

[28] Marc Friedman, Alon Y. Levy, and Todd D. Millstein. Navigational plans for data integration. In *AAAI/IAAI*, pages 67–73, 1999.

[29] Hector Garcia-Molina, Yannis Papakonstantinou, D. Quass, Anand Rajarman, Y. Sagiv, Jeffrey Ullman, Vasilis Vassalos, and Jennifer Widom. The tsimmis approach to mediation: Data models and languages. *Intelligent Information System*, 8(2), 1997.

[30] Joseph Goguen. An introduction to algebraic semiotics, with applications to user interface design. In Chrystopher Nehaniv, editor, *Computation for Metaphors, Analogy and Agents*, pages 242–291. Springer, 1999. Lecture Notes in Artificial Intelligence, Volume 1562.

[31] Joseph Goguen. Social and semiotic analyses for theorem prover user interface design. *Formal Aspects of Computing*, 11:272–301, 1999. Special issue on user interfaces for theorem provers.

[32] Joseph Goguen. Fun with BOBJ, 2002. `www.cs.ucsd.edu/groups/tatami/bobj/`.

[33] Joseph Goguen. Semiotic morphisms, representations, and blending for interface design. In *Proceedings, AMAST Workshop on Algebraic Methods in Language Processing*, pages 1–15. AMAST Press, 2003. Conference held in Verona, Italy, 25–27 August, 2003.

[34] Joseph Goguen. Data, schema and ontology integration. In *Proceedings, Workshop on Combinination of Logics*, pages 21–31. Center for Logic and Computation, Instituto Superior Tecnico, Lisbon, Portugal, 2004.

[35] Joseph Goguen. User interface design class notes, 2006. The CSE 271 website, at `www.cs.ucsd.edu/users/goguen/courses/271/`.

[36] Joseph Goguen and Rod Burstall. Introducing institutions. In Edmund Clarke and Dexter Kozen, editors, *Proceedings, Logics of Programming Workshop*, pages 221–256. Springer, 1984. Lecture Notes in Computer Science, Volume 164.

[37] Joseph Goguen and Fox Harrell. Foundations for active multimedia narrative: Semiotic spaces and structural blending, 2004. To appear in *Interaction Studies: Social Behaviour and Communication in Biological and Artificial Systems*.

[38] Joseph Goguen and Kai Lin. Web-based support for cooperative software engineering. *Annals of Software Engineering*, 12:25–32, 2001. Special issue for papers from a conference in Taipei, December 2000.

[39] Joseph Goguen and Grant Malcolm. *Algebraic Semantics of Imperative Programs*. MIT, 1996.

[40] Joseph Goguen, Guilian Wang, Young-Kwang Nam, and Kai Lin. Abstract schema morphisms and schema mapping generation. Technical report, Dept. Computer Science and Engineering, UCSD, 2004. Submitted for publication.

[41] Joseph Goguen, Guilian Wang, and Vitaliy Zavesov. Schema mapping tool interface design principles. to be submitted.

[42] Georg Gottlob. Computing cores for data exchange: new algorithms and practical solutions. In *Proc. the ACM Symposium on Principles of Database Systems (PODS)*, pages 148–159, 2005.

[43] Georg Gottlob and Alan Nash. Data exchange: Computing cores in polynomial time. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, 2006. to appear.

[44] David Grossman and Ophir Frieder. *Information Retrieval Algorithms and Heuristics*. Springer, 2004. ISBN 1-4020-3004-5.

[45] Object Management Group. Xml metadata interchange (xmi). `http://www.omg.org/technology/documents/formal/xmi.htm`.

[46] Laura Haas, Renee Miller, B. Niswonger, M.Tork Roth, P.M. Schwarz, and E.L. Wimmers. Transforming heterogeneous data with database middleware: Beyond integration. *Bulletin IEEE Computer Society Technical Committee on Data Engineering*, 1999.

[47] Laura M. Haas, Mauricio Hernandez, Howard Ho, Lucian Popa, and Mary Roth. Clio grows up: From research prototype to industrial tool. In *Proc. SIGMOD*, 2005.

[48] Bin He and Kevin Chen-Chuan Chang. Statistical schema matching across web query interfaces. In *Proc. ACM Intl. Conf. on Management of Data (SIGMOD)*, 2003.

[49] LabBook. Bioinformatic sequence markup language (BSML). `http://www.bsml.org/`.

[50] Meredith Lane, James Edwards, and Ebbe Nielsen. Biodiversity informatics: The challenge of rapid development, large databases, and complex data. In *Proc. 26th VLDB Conference*, 2000.

[51] Alon Levy. The information manifold approach to data integration. *IEEE Intelligent Systems*, 13:12–16, 1998.

[52] Alon Levy. Answering queries using views. *VLDB*, 2001.

[53] Alon Levy, Anand Rajaraman, and Joann Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. VLDB Conference*, 1996.

[54] Suzanna Lewis and Erwin Frise. Genome annotation markup elements (GAME). `http://www.bioxml.org/Projects/game/`.

[55] Wen-Syan Li and Chris Clifton. Semint: A tool for identifying attribute correspondences in heterogeneous databases using neural network. *Data and Knowledge Engineering*, 33(1):49–84, 2000.

[56] Wen-Syan Li, Chris Clifton, and Shu-Yao Liu. Database integration using neural networks: Implementation and experience. *Knowledge and Information Systems*, 2(1):73–96, 2000.

[57] ProteoMetrics LLC and ProteoMetrics Canada Ltd. The biopolymer markup language (BIOML). `http://65.219.84.5/BioML.html`.

[58] Jayant Madhavan, Philip Bernstein, Pedro Domingos, and Alon Halevy. Representing and reasoning about mappings between domain models. In *Proc. 18th National Conf. on Artificial Intelligence (AAAI 2002)*, pages 80–86, 2002.

[59] Jayant Madhavan, Philip Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proc. 27th VLDB Conference*, 2001.

[60] Daniel McCracken and Rosalee Wolfe. *User-Centered Website Development*. Prentice-Hall, 2003. ISBN 0-13-041161-2.

[61] Deborah McGuinness, Richard Fikes, James Rice, and Steven Wilder. The chimaera ontology environment. In *Proc. 17th AAAI*, 2000.

[62] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. ICDE*, 2002.

[63] Sergey Melnik, Erhard Rahm, and Philip Bernstein. Rondo: A programming platform for generic model management. In *Proc. ACM Conf. on Management of Data (SIGMOD)*, 2003.

[64] Renee Miller, Laura Haas, and Mauricio Hernandez. Schema mapping as query discovery. In *Proc. 26th VLDB Conference*, pages 77–88, 2000.

[65] Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. 24th VLDB Conference*, pages 122–133, 1998.

[66] Young-Kwang Nam, Joseph Goguen, and Guilian Wang. A metadata integration assistant generator for heterogeneous distributed databases. In Robert Meersman and Zahir Tari, editors, *Proc. Intl. Conf. on Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems*, volume 2519 of *Lecture Notes in Computer Science*, pages 1332–1344. Springer, 2002.

[67] Young-Kwang Nam, Joseph Goguen, and Guilian Wang. A metadata tool for retrieval from heterogeneous distributed XML documents. In P.M.A. Sloot et al., editors, *Proceedings, International Conference on Computational Science*, pages 1020–1029. Springer, 2003. Lecture Notes in Computer Science, volume 2660.

[68] Alan Nash, Phil Bernstein, and Sergey Melnik. Composition of mappings given by embedded dependencies. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, 2005.

[69] Svetlozar Nestorov, Serge Abiteboul, and Rajeev Motwani. Inferring structure in semistructured data. In *Proc. of the Workshop on Management of Semistructured Data*, 1997.

[70] Svetlozar Nestorov, Serge Abiteboul, and Rajeev Motwani. Extracting schema from semistructured data. In *Proc. SIGMOD*, pages 295–306, 1998.

[71] Svetlozar Nestorov, Jeffrey D. Ullman, Janet L. Wiener, and Sudarshan S. Chawathe. Representative objects: Concise representations of semistructured, hierarchial data. In *Proc. ICDE*, pages 79–90, 1997.

[72] Natalya Fridman Noy and Mark A. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proc. AAAI*, 2000.

[73] Natalya Fridman Noy and Mark A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *Proc. Workshop on Ontologies and Information Sharing at IJCAI*, 2001.

[74] Natalya Fridman Noy and Mark A. Musen. Promptdiff: A fixed-point algorithm for comparing ontology versions. In *Proc. the Nat. Conf. on Artificial Intelligence (AAAI)*, 2002.

[75] L. Palopoli, D. Sacca, and D. Ursino. Semi-automatic, semantic discovery of properties from database schemes. In *Proc. the Int. Database Engineering and Applications Symposium (IDEAS)*, pages 242–253, 1998.

[76] L. Palopoli, G. Terracina, and D. Ursino. The system dike: towards the semi-automatic synthesis of cooperative information systems and data warehouses. In *Proc. the ADBIS-DASFAA Conf.*, 2000.

[77] Christine Parent and Stefano Spaccapietra. Issues and approaches of database integration. *Communications of the ACM*, 41(5):166–178, 1998.

[78] Charles Saunders Peirce. *Collected Papers*. Harvard, 1965. In 6 volumes; see especially Volume 2: Elements of Logic.

[79] Deana Pennington, Tony Fountain, Guilian Wang, and John Vande Castle. Spatio-temporal data mining of remotely sensed imagery for ecology. In *GIScience*, 2002.

[80] Lucian Popa, Mauricio Hernandez, Yannis Velegrakis, Renee Miller, Felix Naumann, and Howard Ho. Mapping xml and relational schemas with clio. Demo at ICDE 2002.

[81] Lucian Popa, Yannis Velegrakis, Renee Miller, Mauricio A. Hernandez, and Ronald Fagin. Translating web data. In *Proc. 28th VLDB Conf.*, 2002. Hongkong, China.

[82] Erhard Rahm and Philip Bernstein. On matching schemas automatically. Technical report, Dept. Computer Science, Univ. of Leipzig, 2001. `http://dol.uni-leipzig.de/pub/2001-5/en`.

[83] Erhard Rahm and Philip Bernstein. A survey of approaches to automatic schema match. *VLDB Journal*, 2001.

[84] Rami Rifaieh1, Uddam Chukmol, and Nabila Benharkat. A matching algorithm for electronic data interchange. In Christoph Bussler and Ming-Chien Shan, editors, *Technologies for E-Services: 6th International Workshop*. Springer-Verlag GmbH, 2005. Lecture Notes in Computer Science 3811/2006.

[85] Ferdinand de Saussure. *Course in General Linguistics*. Duckworth, 1976. Translated by Roy Harris.

[86] Mayssam Sayyadian, Yoonkyong Lee, AnHai Doan, and Arnon S. Rosenthal. Tuning schema matching software using synthetic scenarios. In *Proc. of VLDB*, pages 994–1005, 2005.

[87] John Schnase, Judy Cushing, et al. Information technology challenges of biodiversity and ecosystems informatics. *Information Systems*, 28:339–345, 2003.

[88] Dennis Shasha and Kaizhong Zhang. Approximate tree pattern matching. In *Pattern Matching Algorithms*, pages 341–371. Oxford University Press, 1997.

[89] Ben Shneiderman. *Designing the User Interface*. Addison Wesley, 1998. Third edition.

[90] Stylus Studio. Xml schema mapping. `http://www.stylusstudio.com/xsd_to_xsd.html`.

[91] Hong Su, Harumi A. Kuno, and Elke A. Rundensteiner. Automating the transformation of xml documents. In *Web Information and Data Management*, pages 68–75, 2001.

[92] Jeffrey Ullman. Information integration using logical views. In *Proc. Internatinal Conference on Database Theory (ICDT)*, pages 19–40, 1997.

[93] Guilian Wang and Naibin Bai. Study on qsar for general pollutants in organic industrial waste. *Toxic Substance Mechanisms*, 16(4), 1997.

[94] Guilian Wang and Naibin Bai. Structure-activity relationships for rat and mouse ld50 of miscellaneous alchohols. *Chemosphere*, 36(7):1475–1483, 1998.

[95] Guilian Wang and Joseph Goguen. Analysis for schema matching tool user interface. Technical report, Dept. Computer Science and Engineering, UCSD, July 2004.

[96] Guilian Wang, Joseph Goguen, Young-Kwang Nam, and Kai Lin. Critical points for interactive schema matching. In Jeffrey Xu Yu, Xuemin Lin, Hongjun Lu, and YanChun Zhang, editors, *Advanced Web Technologies and Applications*, pages 654–664. Springer, 2004. Lecture Notes in Computer Science, 3007, Proc. Sixth Asia Pacific Web Conf, Hangzhou, China.

[97] Guilian Wang, Yonglong Lu, and Jian Xu. Application of gis technology in chemical emergency response. *Journal of Environmental Sciences*, 11(4):272–278, 1999.

[98] Simon White. How to strike a match. `http://www.catalysoft.com/articles/StrikeAMatch.html`.

[99] Li Xu and David Embley. Using domain ontologies to discover direct and indirect matches for schema elements. In *Proc. Semantic Integration Workshop*, 2003.

[100] Ling Ling Yan, Renée J. Miller, Laura M. Haas, and Ronald Fagin. Data-driven understanding and refinement of schema mappings. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(2):485–496, 2001.