Kanata: Adaptation and Evolution in Data Sharing Systems*

Periklis Andritsos Ariel Fuxman Anastasios Kementsietsidis

Renée J. Miller Yannis Velegrakis

Department of Computer Science
University of Toronto
Toronto, ON, Canada
{periklis,afuxman,tasos,miller,velgias}@cs.toronto.edu

ABSTRACT

In Toronto's Kanata project, we are investigating the integration and exchange of data and metadata in dynamic, autonomous environments. Our focus is on the development and maintenance of semantic mappings that permit runtime sharing of information.

1. INTRODUCTION

Data sharing systems permit the transformation, integration, and exchange of data that has been designed and developed independently. The often subtle and complex interdependencies within data can make the creation, maintenance, and comprehension of such systems quite challenging. We have available a robust arsenal of tools and mechanisms for reconciling semantic differences in how data is represented including views, mappings, and transformation languages. There is also a growing maturity in our knowledge of how to create [19, 20] and use these mechanisms in such tasks as query answering [13], data exchange [9], data integration [17], and data sharing [15].

Given this solid foundation in the tools and modeling structures needed for data sharing, in this project, we are investigating the problem of maintaining such systems. We view an integration system as a dynamic structure which must be maintained and evolved in response to changes in the environment. Such changes may occur in the data or in the metadata (including in the schema and constraints of the data). Evolution may also happen as the integration is developed incrementally over time. Initially, light-weight data sharing mechanisms may be put in place if data administration resources are scarce. As the system is used or requirements change, these mechanisms may be evolved to provide closer data coordination.

In modern data sharing systems, participants rely on one another for service, blurring the distinction between clients and servers, sources and mediators, local and global. We will use the term *peer* to refer to data sources since we are agnostic about the role of the source as a provider or consumer of data. In such systems, autonomy is paramount and decentralized coordination a necessity. As a result, there may be no common global view of data. Rather each peer may define its own view of shared data – a view that may be inconsistent with the view of other peers.

While a single well-designed system may ensure that its own data and metadata evolve in a coordinated fashion, we cannot make such guarantees in a data sharing system. Our focus is on the maintenance of the metadata necessary to achieve semantic integration and sharing of data. Because of this emphasis, we will also be concerned with the problem of understanding if there is a semantic mismatch between the data and metadata.

In the Kanata project, we are developing an integration and transformation process that supports maintainability and adaptability. To achieve this, we are addressing the following research challenges.

- How can integration artifacts, particularly mappings, be adapted? An important challenge is how to maintain mappings in dynamic, autonomous environments where not only the data, but also the schemas and metadata of data sources, may change independently.
- How can automated tools enhance and support the process of developing and evolving data sharing systems?
- How can we ensure the semantic integrity of an integration system? As part of this, we are studying how metadata which is potentially inconsistent with respect to the data can be used.

Our philosophy throughout the project is to provide automated solutions that help users to understand and verify the semantics of the data sharing system. In the next section, we describe in more detail the mappings used in Kanata. In Sections 2.1 and 2.2, we overview our solutions for maintaining two types of mappings: schema mappings and mapping tables. In Section 3, we consider how mappings are used to share data and to support the adaptation process. The paper concludes in Section 4.

2. MAPPINGS IN KANATA

Before introducing our solutions for maintaining and adapting metadata, we briefly describe some of the common types of metadata that have proven valuable in data sharing. Our overview is meant to be exemplary, rather than exhaustive.

A fundamental requirement for all data sharing tasks is the establishment of associations between data represented in different ways. Since our focus is on supporting structured querying, we consider mappings that permit the translation of structured queries over heterogeneous data representations. Such mappings depend invariably on schemas.

*Funding provided by NSERC, CITO, and a Premier's Research Excellence Award.

aid	*city*
YYZ	Toronto
LAX	Los Angeles
JFK	New York

(a) AC_Airports

pid	*type*	*cap.*
B747	Boeing	438
B767	Boeing	305
A320	Airbus	300

(b) AC_Aircrafts

fno	*from*	*to*	*date*	*time*	*pid*
AC401	YYZ	LAX	09/09	18:00	B747
AC541	YYZ	JFK	09/15	10:30	B767
AC543	YYZ	JFK	09/21	12:00	A320
AC548	YYZ	JFK	09/23	10:30	A320

(c) AC_Flights

flight	*dep*	*dest*	*date*	*time*
UA134	JFK	SFO	09/14	17:00
UA120	JFK	LAX	09/15	10:30
UA203	YYZ	JFK	09/15	10:30
UA208	YYZ	JFK	09/21	12:15

(d) UA_Flights

Figure 1: Air Canada and United instances

The presence of a schema permits the more sophisticated structured queries that are not possible over unstructured, schema-less data. We refer to mappings that permit the translation of structured queries as *schema mappings*.[1]

The schema mappings we consider include traditional views (both global-as-view and local-as-view mapping systems) along with more general mappings relating the schemas of independent data sources. In their simplest form, mappings are queries specifying how data from one peer may be transformed into the structure of another. However, mappings may also specify general constraints on the data that is to be shared between peers. Our work focuses on maintaining schema mappings, including mappings that have been written manually, generated with the help of automated tools, or produced as the output of an integration or schema transformation process.

In addition, we consider the use of *mapping tables* which are data-level mappings that record the association of values in different sources [15]. Even when two peers share common schemas, they may not share common vocabularies for data values. Identifiers may be drawn from different domains, or the lexicon used for a descriptive attribute may differ. Mapping tables are complementary to schema mappings. Schema mappings and mapping tables may be used alone or in concert to translate queries and share data. However, as we show in the next sections, different techniques may be required to maintain them.

2.1 Schema-level Mappings

When the schema of a source is modified some of the mappings that have been defined on them may need to be updated or adapted to ensure that they meet the requirements of the modified schema. We refer to this process as *mapping adaptation* to differentiate it from view adaptation [12], view maintenance [24] and view synchronization [16]. View adaptation and maintenance consider ways of updating the instance of a materialized view when there are changes to the base data or the view definition, respectively. On the other hand, view synchronization, like mapping adaptation, considers ways of updating the view or mapping definition when there are changes to the schemas, but mapping adaptation deals with a richer set of mappings and schema changes.

Despite the many interesting results that exist in data translation, mapping adaptation still remains a mostly manual task that is performed by data administrators who are familiar with the semantics of the schemas, and the transformation language. A data administrator visits the existing mappings, one at a time, and rewrites those that are found to be affected by the schema changes. This task is laborious, time consuming and error-prone. Alternatively, one could drop existing mappings and generate new ones for the modified version of the schemas, an effort that can be aided tremendously by modern tools that suggest schema matches (which associate schema elements) [21] and schema mappings (which associate schema instances) [19, 20]. Or one could generate new mappings between the original and modified schemas, and compose these mappings with existing mappings, using newly emerging solutions for mapping composition [6, 18, 10]. However, with the latter approaches, the mapping generation does not consider past user input, specifically that the affected mappings may have been based on input from human experts. It is the semantic decisions input by these experts that we would like to preserve in order to save the most precious administrative resource, human time.

To address this problem, we have developed a comprehensive framework and implemented a mapping adaptation tool [22] in which a designer or administrator can change and evolve schemas and mappings. The tool detects mappings that are made inconsistent by a schema change and incrementally modifies the mappings in response. This approach has the advantage that we can track semantic decisions made by a designer either at the time the mappings were created, or as they are evolved. These semantic decisions are needed because schemas are often ambiguous (or semantically impoverished) and may not contain sufficient information to make all mapping choices. We can then reuse these decisions when appropriate.

Consider, for example, the case where we need to populate a relation *T(city,type)* with data retrieved from the Air Canada source (Figure 1). Relation T contains the various cities and for each city, the type of aircrafts that land at its airports. The query that retrieves this information and generates data in T is the following.

(1) SELECT *a.city, p.type*
FROM *AC_Airports a, AC_Flights f, AC_Aircrafts p*
WHERE *a.aid=f.from* AND *f.pid=p.pid*

Assume now that for some reason the owner of the Air Canada data source decides to remove the attribute *pid* of relation *AC_Flights*. This will make the above mapping invalid since this attribute is used in the join between *AC_Flights* and *AC_Aircrafts*. As a solution, our automatic adaptation algorithm will detect this, and will replace the specific invalid mapping with the following two rewritings.

(2) SELECT null, *p.type*
FROM *AC_Aircrafts p*

[1]Note that in the literature, schema mappings are often just called mappings [17]. We use the term *schema mappings* to distinguish them from mapping tables.

(3) SELECT *a.city, null*
FROM *AC_Airports a, AC_Flights f*
WHERE *a.aid=f.from*

The first mapping generates tuples from the *type* values in *AC_Aircrafts* but sets the *city* attribute to null, while the second generates tuples from the values of attribute *city* but sets the *type* attribute to null. The reason is that our algorithm was able to detect the fact that in the modified schema, aircraft types are not semantically related to airport cities since there is no join path between the respective attributes to associate their values. On the other hand, it can be seen that the second mapping, although it does not retrieve any values from relation *AC_Flights*, it has preserved the join between relations *AC_Flights* and *AC_Airports*. This is a consequence of one of the main principles of our approach, to preserve as much as possible the semantics of the initial mapping. Since the initial mapping retrieved *city* values that were related to tuples in *AC_Flights* through attribute *from*, the rewriting preserves this join.

Another advantage of our tool is that it can detect when mappings are affected not only by structural changes but also by changes to the semantics of the schemas. Consider, for example, the case where we have the Air Canada schema of Figure 1 and the mappings (2) and (3) presented above, and assume that a new foreign key constraint is added from attribute *pid* of *AC_Flights* to attribute *pid* of *AC_Aircrafts*. Our algorithm is able to detect that a new join path has been created between *city* and *type* attributes and suggest a rewritten mapping that generates more semantically complete information (i.e., for each city, it also finds the associated types of aircrafts that land at the airports of that city). In other words, Mapping (1) above is generated.

In some cases, the algorithm may generate more than one mapping. In order to assist the user even further in dealing with the evolving nature of the integration system, we have developed a metric to measure the semantic similarity between two mappings [23]. The metric is based on the number of common relations and conditions that are involved in the mappings, compared to the relations and conditions that are not shared by the mappings. This metric is used to rank the candidate rewritings.

For example, Mapping (3) is considered semantically closer to Mapping (1) than to Mapping (2), but semantically further from the following mapping.

(4) SELECT *a.city, null*
FROM *AC_Airports a, AC_Flights f*
WHERE *a.aid=f.from* AND *f.to='JFK'*

The reason is that although both Mappings (4) and (3) have the same number of common components, Mapping (4) is less preferable since it contains an additional condition *f.to='JFK'*.

Apart from adapting mappings in response to schema changes, we have to ensure that data translation at the value level is also consistent with the evolution of the system, and this is the issue we deal with next.

2.2 Data-level Mappings

Mapping tables are data-level mappings that record the association of values in different sources [15, 5]. An example of such a table is shown in Figure 2 which lists the associations between the codes of code-share flights of the AC and UA airlines. As the contents of the sources, and thus the stored values, change over time, mapping tables should be evolved to reflect these changes. In our airline example, new flights can be constantly added to the corresponding airline databases. If some of these new flights are code-share flights, the corresponding mapping table must be updated to reflect this fact. The evolution of mapping tables involves both discovering new mapping tables between sources and updating the value associations recorded in existing tables. In the following paragraphs, we review techniques for both types of evolution.

MappingTable

fno	flight
AC541	UA203
AC543	UA208

Figure 2: An example mapping table

2.2.1 Discovering Mapping Tables

In this section, we present a way of discovering duplicate or associated tuples that exist in different sources. These duplicates can be identified by different values. In our example, the different airlines may use different identifiers for the same flight. This information may be entered manually into the system, but often it may be missing or incomplete. We are studying the problem of developing automated tools that help to discover or suggest new associations among values. These discovered associations or aliases may be used to populate mapping tables. For example, in the case of *AC_Flights* and *UA_Flights* sources of Figure 1 our task may be to identify code-share flights from these two sources.

Generally speaking, if we seek to characterize duplication in a set of tuples, we need a way of measuring their similarity. Numerous de-duplication and record linkage approaches exist, but many of these require domain-specific similarity measures. In Kanata, we use an information-theoretic measure of similarity, that of *Information Loss*, used in the LIMBO clustering algorithm [3] for categorical data and in a set of tools for structure discovery [2]. Intuitively, this similarity measure deems two tuples to be similar if the information in the values contained in the tuples is similar. Using information loss, the similarity measure does not require knowledge of application specific measures or of orderings on values which may not be shared between different sources.

In the current setting, we first need some way of combining the information (tuples) that exists in the different sources. In the airline example of Figure 1, we need to combine the flights represented by the different sources. If the set of values that appear in the individual sources intersect, then we expect that the overlap, and thus the similarity of the values in the tuples, increases. This is the case for the flights AC541 in *AC_Flights* and UA203 in *UA_Flights*. We have introduced a clustering-based approach for suggesting possible duplicate tuples [2]. We first perform one pass over the data and build summaries, or more precisely, representative tuples that represent similar groups of tuples. These summaries serve as guidelines for assessing the similarity of tuples; we build summaries and then assign tuples from different relations to these summaries. More formally, we identify duplicate tuples as follows.

1. Given a set of sources $S_1, \ldots, S_k$, determine a set of related descriptive attributes.

2. Pick an accuracy threshold for the summaries.

3. Using information loss as a distance measure, discover the summaries that best reflect the original data.

4. Assign tuples from the original data set to summaries.

5. Identify potential duplicates by inspecting the tuples assigned to the same summaries.

Our technique identifies whole tuples as duplicates, suggesting that their identifiers may be aliases. For example, our technique identifies the following pairs of flight numbers as identifying duplicate tuples: (AC541,UA203) and (AC543,UA208). These identifiers can populate a new or an already existing mapping table such as the one in Figure 2. Our goal is to help a curator to find duplicates across sources or to find duplicates within a single dirty (or perhaps integrated) source.

Some interesting points that must be made are the following.

- In the aforementioned technique, we assumed that the domains of the individual sources must overlap in order to discover duplicates. This assumption may be relaxed by having q-grams of values instead of the whole values. A q-gram is the set of all q-sized subsets of consecutive characters in a string.

- In order to represent the tuples of individual sources more accurately in our representation, we may make use of already existing mapping tables. For example, if a mapping between the values AC541 and UA203 exists, then we can use either one of the values in the new representation instead of both values.

2.2.2 Reasoning on Mapping Tables

In the previous section, we use the contents of the sources in order to drive the discovery of new tables or augment the recorded associations in existing tables. That is, we use data to discover new meta-data. In the following paragraphs, we perform similar operations by relying solely on mapping tables. That is, we use existing meta-data to produce new meta-data. In turn, this new meta-data might lead to the production of new data. This feedback loop between data and meta-data is one of the main characteristics of our approach and is a recurrent theme which we encounter during the process of query answering in Section 3.

As mentioned, the techniques presented in this section rely on existing mapping tables either to discover new mapping tables or to augment the associations recorded in them. Furthermore, we present here a technique that can be used to check the validity of both the newly created tables and the additionally inferred associations. The key idea, underlying all techniques, is the treatment of mapping tables as constraints on the association of tuples between sources. Although we do not introduce formally the notion of mapping constraints (see [15] for a formal introduction), we present an example of how mapping tables can be used as constraints. For this, consider the mapping table for code-share flights shown in Figure 2. Given this table, from the three AC flights from YYZ to JFK, in Figure 1, only the first two AC flights are associated with the two corresponding UA flights between the same origin and destination. Any other association of tuples, from the two relations, does not satisfy the mapping table m.

Research in database constraints focuses on addressing two main problems, namely, the inference and consistency problems [1]. In the context of mapping constraints, a solution to the inference problem can be used either to discover new mapping tables from a network of existing ones or to augment the associations that are recorded in existing tables. As an example, suppose that there is a mapping table for code-share flights between the UA and Lufthansa airlines. Our techniques can use this mapping table, plus the one in Figure 2, to discover if there are any code-share flights between the AC and Lufthansa airlines.

A solution to the consistency problem of mapping constraints offers a way to validate or verify the associations of values that are discovered either through our inference techniques or through the techniques described in the previous section. As an example, we assume that through one of the techniques, we are given the following additional information.

fno	flight
AC401	UA120
AC541	UA203
AC543	UA208

Suppose we have a mapping indicating that if a *fno* X is associated with a *flight* Y, then these two flights must originate from the same city.[2] Using our solutions for the consistency problem, we are able to detect that the first tuple above is inconsistent. The user can be notified of the inconsistency and she can provide a strategy to resolve it. Our system allows the user to specify a strategy so that future inconsistencies between the two tables are automatically resolved.

In addition to providing solutions for the aforementioned problems, our work has to deal with the technical difficulty of addressing these problems in a setting where the available constraints are not physically located in one place, that is, in one source, but they are distributed over a networked of sources. As such, our solutions are customized in order to work in a dynamic, networked environment.

3. DATA SHARING IN KANATA

In this section, we show how mappings are used to support data sharing between different sources. Since data sharing is performed by means of querying, we present here techniques to query the data using the mappings and to query the mappings themselves. In Kanata, both the data and metadata evolve and thus both types of queries are useful.

3.1 Querying Mappings

Data coming from the integration of independent, physically distributed, heterogeneous sources that may have been developed with different requirements in mind, are not always well-understood and accepted. Some of the original data semantics may be lost during the integration and, as a consequence of the transparent access provided by the integration, the notion of distinct sources and their parts often

[2]A similar mapping is illustrated more explicitly in the next section.

disappears from queries and results. Hence, searching in an integrated manner for data that is not only relevant but also best suited to a task at hand, is a difficult process. In addition, there are numerous applications where users need to know the origin of the data and reason about it in order to evaluate the quality of the retrieved results. Knowing how and from where each particular data element was derived allows users to apply their own judgment to the credibility of that information and decide whether some particular data is a semantically correct answer to their query. In systems where information from multiple sources is used, such knowledge may assist in interpreting the data semantics and resolving potential conflicts among data retrieved from different sources. In several emerging applications the ability to analyze "what-if" scenarios in order to reason about the impact of the data coming from specific sources (or specific parts of them) is of paramount importance.

Furthermore, in dynamic environments where mappings and schemas may change frequently, the semantics of the integrated data may also be changing. In such cases, it would be helful to be able to query not only the data, but also the meta-data information, that is, schemas and transformations (i.e., mappings). For that reason, we have elevated schemas and mappings to first class citizens.

As a first step towards this direction we have developed and implemented a representation model for schemas and mappings. This meta-data information can then be queried the same way regular data is queried.

As an example, consider the case where a user does not understand the exact semantics of the data of a specific schema attribute. She can pose a query that requests the mappings populating the particular attribute, as well as the parts of the remote schemas that contribute data to this attribute. The returned result may provide the user with the required information to better understand the semantics of the individual element. For example, attribute ts in a table Person may not be well-understood. A query on what mappings are using this attribute may return a mapping that specifies that a value in ts is generated by multiplying the values of attribute $monthlySalary$ by 12. Viewing this, a user may come to understand the meaning of ts to be the yearly income.

In many cases, it is also important to determine for a specific data element in an integration where it originates and through what mapping. To keep track of this information we have introduced annotations on data values in order to associate them with their meta-data information. Annotations are not simply super-imposed information but can be queried along with data. We have also developed an extended query language that offers the capability of uniformly manipulating data and meta-data by utilizing the proposed representation model and data annotations.

3.2 Querying using Mapping Tables

Our framework for query answering, through mapping tables, assumes that a user poses queries only with respect to its local source. The motivation for this assumption is to free the user from the requirement to be fully knowledgable, or even aware, of the schemas of other sources. Given a user query Q, it is the responsibility of the system to translate this query to one that can be executed over the schemas of the sources that are integrated with the current user source. For this, we provide a rewriting mechanism that uses mapping tables to translate query Q to a set of queries that can be executed over the integrated sources. Although the idea of query translation is not novel, the context in which it is applied is. Traditionally, query translation relies on the use of schema mappings that specify how the schemas of the sources are related. During the query translation, these relatively small expressions are used. Mapping tables however contain data and may be very large. Query translation in our environment involves manipulating these large tables.

In more detail, our work shows how mapping tables can be used to translate select-project-join queries, where the selection formula is *positive*, i.e., it has no negation and it consists of conjunctions and disjunctions of atoms of the form $(A = B)$ and $(A = a)$, where A and B are attribute names and a is a constant. We consider both sound translations (which only retrieve correct answers) and complete translations (which retrieve all correct answers, and no incorrect answers). In this setting, the complexity of testing for sound translations is Π_2^p-complete, in the size of the query. Since large queries rarely occur in practice, the high complexity is not an obstacle. Our experimental evaluation of the algorithm indicates that it works efficiently in practice [14]. We also propose and implement algorithms for computing sound and complete translations, and we offer experimental results that show the efficiency of these algorithms.

3.3 Querying Inconsistent Information

In static environments, data sources are populated only after the schemas and mappings have already been designed. Therefore, it is possible to ensure, at every point in time, that data remains consistent. In contrast, in a dynamic environment, data sources may be populated at the moment that the schemas or mappings change. Then, the question that arises is: what shall we do with a source if, as a result of a change, it becomes inconsistent?

As a simple example, suppose that a design decision has been made that the flight time information between Air Canada and United Airlines should be synchronized. This can be specified with the following schema mapping:

$$MappingTable(fno, flight) \wedge AC\_Flights(fno, date, time) \wedge UA\_Flights(flight, date', time') \rightarrow (date = date' \wedge time = time')$$

Notice that the mapping is not a GLAV mapping [11], since it involves relations from two different sources on its left-hand side. Also, the mapping table plays an important role here in order to relate the flights of the two companies.

It is easy to see that *AC_Flights* and *UA_Flights* are inconsistent with respect to the new mapping. For instance, codes UA208 and AC543 stand for the same flight, which departs at 12:00 according to Air Canada, and at 12:15 according to United. If a query asks for the departure time of the flight, what should be the answer?

The traditional approach to deal with these situations is *data cleaning* [7]. In our example, this amounts to deciding whether United or Air Canada has the correct departure time for the flight. Data cleaning techniques are often not applicable in our context. First, because they are semi-automatic, and the cost in terms of human involvement may become prohibitive when the cleaning has to be done every time the metadata changes. Second, because sources are autonomous, and may therefore refuse to be "cleaned" just because of changes in the mappings.

Our approach resorts to run-time reconciliation of inconsistencies, drawing upon the notion of *repairs* originally developed in the area of consistent query answering [4]. A repair is an instance that minimally differs from the inconsistent database under set inclusion. Repairs need not be unique; each repair corresponds to a possible "cleaned" instance. A *consistent answer* is an answer that appears in *every* repair of the database. A *possible answer* is an answer that appear in *at least one* repair of the database.

The potentially large number of repairs leads us to ask whether we can compute the consistent answers of q efficiently. The answer to this question is known to be negative in general [8]. However, this does not preclude the fact that there may exist large classes of queries for which the consistent answers can be retrieved efficiently. In particular, we have identified a large and practical class of conjunctive queries (i.e., queries that use selection, projection, and join) that enjoy this property. For this class, we have given query rewriting algorithms that, given a query q, produce a query q' that retrieves the consistent answers *directly* from the inconsistent database.

The consistent query answering module of Kanata will use schema mappings and mapping tables in order to retrieve the possible and consistent answers to data sources. Notice that the approach of providing consistent answers is very conservative, in the sense that a tuple is not in the answer unless it appears in every repair. If a user is not satisfied with the consistent answer to a query, our tool would provide her with the possible answers, and an explanation of what mappings make specific answers inconsistent.

4. CONCLUSION

In this paper, we presented a system, called Kanata, that manages the evolution of data sharing systems. A distinctive feature of Kanata is that it introduces techniques that exploit the interplay between the data and metadata levels of the system. In order to manage system evolution, metadata-level techniques are used to automatically update mappings in response to schema changes; data-level techniques are used to allow query answering even when some constraints of the system are violated. In order to discover associations between data sources, data mining techniques are used at the data level, and inference techniques are used at the metadata level. Also, metadata information can be queried in the same way as any other data in the system.

5. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] P. Andritsos, R. J. Miller, and P. Tsaparas. Information-Theoretic Tools for Mining Database Structure from Large Data Sets. In *ACM SIGMOD Int'l Conf. on the Management of Data*, pages 731–742, 2004.

[3] P. Andritsos, P. Tsaparas, R. J. Miller, and K. Sevcik. LIMBO: Scalable Clustering of Categorical Data. In *Advances in Database Technology - Int'l Conf. on Extending Database Technology (EDBT)*, pages 123–146, 2004.

[4] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 68–79, 1999.

[5] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, and J. Mylopoulos. The Hyperion Project: From Data Integration To Data Coordination. *ACM SIGMOD Record*, 32(3):53–58, 2003.

[6] P. A. Bernstein, A. Y. Halevy, and R. Pottinger. A Vision of Management of Complex Models. *ACM SIGMOD Record*, 29(4):55–63, 2000.

[7] M. Bouzeghoub and M. Lenzerini. Introduction to the special issue on data extraction, cleaning and reconciliation. *Information Systems*, 26(8):535–536, 2001.

[8] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 260–271, 2003.

[9] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *Proc. of the Int'l Conf. on Database Theory (ICDT)*, pages 207–224, 2003.

[10] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 83–94, 2004.

[11] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proc. of the National Conf. on Artificial Intelligence (AAAI)*, pages 67–73, 1999.

[12] A. Gupta, I. Mumick, and K. Ross. Adapting Materialized Views After Redefinition. In *ACM SIGMOD Int'l Conf. on the Management of Data*, pages 211–222, 1995.

[13] A. Y. Halevy. Answering Queries Using Views: A Survey. *The Int'l Journal on Very Large Data Bases*, 10(4):270–294, 2001.

[14] A. Kementsietsidis and M. Arenas. Data sharing through query translation in autonomous systems. *To Appear in VLDB*, 2004.

[15] A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *ACM SIGMOD Int'l Conf. on the Management of Data*, pages 325–336, 2003.

[16] A. J. Lee, A. Nica, and E. A. Rundensteiner. The EVE Approach: View Synchronization in Dynamic Distributed Environments. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):931–954, 2002.

[17] M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[18] J. Madhavan and A. Halevy. Composing Mappings Among Data Sources. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 572–583, 2003.

[19] R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 77–88, Cairo, Egypt, September 2000.

[20] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating Web Data. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 598–609, August 2002.

[21] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The Int'l Journal on Very Large Data Bases*, 10(4):334–350, 2001.

[22] Y. Velegrakis, R. J. Miller, and L. Popa. Mapping adaptation under evolving schemas. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 584–595, 2003.

[23] Y. Velegrakis, R. J. Miller, and L. Popa. On Preserving Mapping Consistency under Schema Changes. *The Int'l Journal on Very Large Data Bases*, 2004. Submitted.

[24] J. Widom. Research Problems in Data Warehousing. In *International Conf. on Information and Knowledge Management*, pages 25–30, Baltimore, Maryland, 1995.