Pragmatics and Open Problems for Inter-schema Constraint Theory

Arnon Rosenthal, Len Seligman
The MITRE Corporation
{arnie, seligman}@mitre.org

Abstract

We consider pragmatic issues in applying constraint-based theories (such as that developed for data exchange) to a variety of problems. We identify disconnects between theoreticians and tool developers, and propose principles for creating problem units that are appropriate for tools. Our Downstream Principle then explains why automated schema mapping is a prerequisite to transitioning schema-matching prototypes. Next, we compare concerns, strengths, and weaknesses of database and AI approaches to data exchange. Finally, we discuss how constrained update by business processes is a central difficulty in maintaining n-tier applications, and compare the challenges with data exchange and conventional view update.

1. Introduction

We examine pragmatics of applying constraint-based theories (such as that developed for data exchange) to a variety of problems. We identify barriers to employing the results of the theory as is, and ask what repackaging and additional results would be helpful for a variety of problems. In doing so, we assess the needs and predilections of several communities who ought to be consumers of the theory:

- Developers of direct database-to-database exchange tools, both matchers and mappers
- Data mediation researchers and developers in the AI community
- Creators of update methods and processes for business objects, a task that can involve both view definitions and constraints on the base tables.

Data exchange theory has provided very useful insights for builders of schema mapping and integration tools. For years, tool researchers had sought to find matching attributes across schemas [RaBe01], or to reuse known matches [RoSe94, MRB03]. Data exchange theory, which emerged over the past five years (extending Local As View), complements these heuristics by asking "what do the matches mean?" and "what do matches have to do with mappings of instance sets?" It answers these questions by examining the consequences of constraints among loosely coupled schemas for tables or documents.

Data exchange has since become a major topic at conferences, e.g., [Bert05, Ko05]. It is formulated in ways that facilitate stating general research problems and comparing research results -- proper concerns of database theorists. Unfortunately, the research may not be readily accessible to practitioners.

We believe the practical value would increase greatly if the results were repackaged and extended to apply directly to problems addressed by several other potential consumer communities -- and conversely if these development and administration communities factored their work to better exploit theory.

To this end, we identify gaps (in technology or in understanding) that currently discourage technology transfer, but should be easy for theorists to bridge. We consider concerns of tool developers, application developers, AI researchers doing information integration, and providers of business object systems.

Standing somewhere between PODS theorists and commercial system builders, we will attempt to build a bridge that influences agendas and catalyzes progress. Specifically, we:

- Explain important practitioner concerns, including some that theorists could meet easily.
  - Suggest tactics for tweaking theory and systems to obtain a better fit
  - Why it is appropriate for theorists to play a larger role in technology transfer
- State a "Downstream Principle" to clarify why tools that address later stages of the integration process (i.e., mapping) transfer to industry before more "upstream" tools (i.e., matching).
- Examine the tradeoffs between DB-style and ontology-based data exchange.
- Discuss the applications (worth $millions) and challenges of handling *updatable business objects*.

2. Making Data Exchange Theory More Useful

Data exchange theory has recently provided a formal basis for using schema matching knowledge to obtain instance set and query mappings. In particular, it has contributed to IBM Almaden's CLIO system [HMH01].

Yet much of the theory seems difficult to apply. Imagine writing product documentation telling an application

developer that they are seeing the intersection of all "certain" answers to their query, or explaining the concept of a "core" to a domain expert assigned to do integration. Then imagine explaining how to write applications that tolerate these difficulties.

To assist practitioners, we must decompose their problems into a piece that tools can help solve (with theory that supports the tools) and a *residue* problem that requires human intervention. For the tools to be valuable to a user, the residue problem must be

- smaller than the original task
- understandable to the intended user

We also want to exploit humans' strengths. Today, formally intractable problems are routinely solved by people who write programs, based on domain knowledge or finding tractable cases. A fine goal for tools is to do solid formal inference for pieces where it is feasible, so humans do not need to do such tasks (and redo them each time circumstances change).

A residue that is *not* acceptable is to return a data mapping, with a set of caveats about not having a unique minimal certain answer. Few application administrators today understand these concepts. Calls to train them better are futile. Tools should support an ever expanding set of users, empowering domain experts rather than just computer scientists.

One better alternative for data exchange tools is to generate an intermediate schema (e.g., with some constraints omitted) and pose the problem of exchanging data between the intermediate and the desired target. When determining how much to do in one stage, consider splitting at the point where a different human has the needed skills. For example, the person who understands attribute semantics might be different from the one who has the identifier and quality knowledge needed to drive data cleaning [MRS+05].

Another alternative is to formulate the residue as a (smaller) knowledge capture problem. In such cases, you may want to insist that the desired knowledge be phrased in terms of a schema the user intuitively understands—i.e., typically that of the source or target.

We now give some examples of how hard problems might be decomposed so that humans can assist appropriately.

Researchers report that they cannot get a unique solution when there are key constraints on the target but not on the source. For users, the operative explanation is not formal -- it is that distinctions that require real world knowledge cannot be made by generic tools. If a source states two different eye colors for me and the target wants a unique value, of course the theory can't decide which matches the real world. One would not want a theory that gave a unique answer.[1] Instead, theory can help create tools that treat target constraints as heuristics, and help identify which constraints to defer. That is, one would generate a data exchange mapping to a schema that omits the objectionable constraints. One would then support humans and other tools (e.g., data cleaning) in dealing with the residue.

Data exchange theory may be profitably combined with view update (Section 5). In particular, both can benefit from having a staging area (intermediate result) that holds only problematic tuples. For many information feeds, atomicity is not vital; one prefers to immediately assimilate correct instances.

We conclude this section with three caveats.

C*ompleteness is nice to have, but is not crucial*. Generally, data is incomplete (we do not know *all* sufferers from disease X), and constructs that prevent completeness nevertheless provide useful knowledge. Applications and business procedure are tolerant of incomplete results.

Administrators are expert only about schemas they use. They lack instinctive feel for properties of an unfamiliar virtual database. Consequently, it may be better to capture their knowledge in terms of familiar relation schemes (or subsets thereof), perhaps abstracting a bit to hide details of value representations (e.g., lengths as feet vs. meters) and constraints. A research challenge is to use source metadata to infer metadata for the intermediate schema. One does not want to tell the user to drill down back to the original to determine an attribute's measurement units. For metadata that cannot be inferred, perhaps it can be measured at run time (e.g., intrinsic data quality estimators such as consistency and completeness).

Enterprise scale systems are unlikely to satisfy your assumptions everywhere. Does that make your techniques inapplicable? Please help practitioners understand how to salvage your techniques, when your assumptions do not apply.

For example, one (valuable) view update paper [Kell86] simplifies the problem as follows: "We consider the problem of updating databases through views composed of selections, projections, and joins of a series of Boyce-Codd Normal Form relations." To which we react: What does this have to do with me, if I don't have a SPJ view of BCNF relations? Suppose I have a more complex query, part of which might be expressed as SPJ? We agree that it is reasonable for research papers to start by describing what they have succeeded in solving. But it is also legitimate research—and a good use of theorists' skills—to advise on how to make the result relevant to a noncompliant problem.

3. The Downstream Principle for tools: Full automation is more valuable than partial

Previous sections identified disconnects between data exchange theory and the requirements of practical tools and described how some of those disconnects might be addressed. We now propose the Downstream Principle, which provides further guidance on research topics that are most likely to have the greatest practical impact. We begin with definitions:

- *Upstream* automation helps integration engineers but does not produce value for others.

[1] Our warning is in the same spirit as [MS89], which observed there was no reason to believe that users' intent was preserved by heuristics to remove ambiguity in "abbreviated" queries.

- *Downstream* automation produces a runnable application that provides value to customers outside the data integration group.

The Downstream Principle asserts that one gets greater value from automating downstream processes than upstream ones. When an upstream task is automated, there remains considerable work to be done. If the subsequent work is not automated, humans need to intervene again before customers receive value. (Customers could be end users or developers of application functionality).

Canal analogy: A canal that connects to the sea will encourage more trade and industry than one that connects two inland points with a footpath to the sea. Even for those who must reach the sea from the farther inland point, it is better to walk first and load into a boat once, rather than load, unload, walk, and load at the sea. In addition, the cost of transport from the start of the canal will be drastically reduced, so new uses may be found from there onward. (Analogy: When it became "free" to create a GUI forms interface, users of all systems got GUIs rather than command line syntax.)

Research prototypes for data exchange mostly support *match*, which automatically identifies likely semantic correspondences, but does not fully specify the semantics of the exchange. Commercial tools support *map*, which produces detailed, executable code that transforms instances of a source schema to those of a target schema. While there has been good and useful research on both match and map, we believe our community's impact will be greatest if knowledge capture is solved for map (i.e., downstream), for the following reasons.

First, *automating the higher skill task results in a bigger savings*. Domain experts can often perform match with minimal assistance from programmers. (The current state-of-the-practice is to perform "data crosswalks" with spreadsheets serving as the knowledge capture tool.) In contrast, most domain experts cannot create code, and so are powerless without help in mapping, which creates runnable code.

Second, *map tools are more useful for maintenance.* The bulk of software costs are for changing existing systems, not for initial deployment. The benefit of a matching tool (upstream) is the difference:

EffortSavedDeterminingMatches – ImportExportEffortForMatchTool

For a small maintenance task, the first term is small and the second becomes much more significant. The likely result is that a match tool would not be purchased or used.

For data exchange, there are scenarios where match is a much smaller problem than map. For example, dozens of Department of Defense systems exchange information about events of interest through a narrow-scope interface called Cursor On Target. Instead of waiting for thorough integration, they base their exchange on a simple Event model consisting of What, When, and Where (with hand-crafted extensions where needed). In essence, they have agreed on a neutral XML schema of ~15 elements to represent Time and Position and to refer to a preexisting "What" standard for describing military entities of interest (e.g., distinguishing land/sea/air, US/friendly/unfriendly, etc.).

In this situation, matching cost is trivial -- a person immediately can recognize the When, Where, and What that is of interest for a particular community. In contrast, mapping is nontrivial; it requires specifying detailed transforms from participants' internal representations to the neutral interface (e.g., across different coordinate systems). Without automation, skills are required to insert and deploy needed code. (Relevant research includes mapping discovery [IMHB04] and "context" mediation of value representations [GBMS99].)

4. Comparing DB and AI Mediation Approaches to Data Exchange

Data exchange researchers and AI researchers pursuing ontology-based data mediation are too frequently unaware of each other's concerns (with notable exceptions, e.g., [DHN04, SE05]). For example, most data exchange theorists seem unaware of the advantages of ontology formalisms, while ontology-driven approaches seem unconcerned with returning the right instance set. We therefore try to explain the differences in perspectives.

Our first comments are about basic assumptions, independent of integration: Database schema formalisms (SQL, XML, ER) center on n-tuples (rather than individual attributes' values) and *sets* of instances. Many people still think of a tuple as a physical record with an upper class accent. The database formalisms thus map naturally upward to a GUI display about the entity, and downward to a record structure that query optimization uses as a unit of physical access. Instance sets, meanwhile, are central to storage, query languages, and optimization, but each schema has just one set. The AI community treats sets as a tertiary issue. As best we can tell, their mediators do not address the issues of certain answers, though other AI work addresses possible worlds.

Database-style data exchange tools typically emphasize connecting systems' schemas directly (e.g., a database to a virtual query interface). This suits a results-oriented market, aligning with incentives (do just enough for your own purpose) and providing relatively "instant gratification" [HEDI03, RoSe94, RSRM01]. In contrast, the typical AI approach says "first capture a model of your problem domain, and then we'll do something useful for you". Database researchers have also discussed use of a neutral conceptual schema (notably in Local as View), but the tools have given less emphasis.

AI data mediation work also dates back to the early 1990s, notably to the Carnot system at MCC [CHS91] and the SIMS system at ISI [ArKn93]. The greatest difference from the database style is that it centers on a neutral domain

model (e.g., an ontology). Also, AI researchers moved earlier to a heavy reliance on logic and inference.

The AI approach is based on ontology formalisms, today typically centering on the Web Ontology Language, or OWL [W3C04]. This formalism provides important advantages:

- First, OWL is a W3C standard with a substantial research community, much freeware, and some existing domain models. (In contrast, database logic approaches, even Datalog, lack influential standards.) In OWL Full, one can map from data to metadata.
- Second, most of the administrator work for both matching and mapping concerns derivations between individual elements (attributes), not whole records. Relational and XML formalisms lack such constructs, and one must employ queries or a non-standard assertion language. In contrast, OWL supports typed relationships directly.
- Third, OWL has an extensible type system that is easily adapted to provide relationships useful for integration. For example, one may assert that Database1.Car "can be understood by" Database2.Vehicle. This relationship is a first cousin of IS-A; it indicates set containment (so the data can be used) but not attribute inheritance across systems. (That is, Database1.Car will not necessarily inherit all of Database2.Vehicle's property types.) Inference is much more powerful because relationships within and across ontologies can employ the same formalism.

Despite OWL's attractive features, DB approaches have several advantages. First, there are far more domain models today expressed in XML (despite the technical difficulties of hierarchies), UML class diagrams, and Entity-Relationship models than in OWL, and database researchers' logics (e.g., CLIO's [HMH01]) are well tuned to XML and relational models. Second, it is unclear which approach is better for transforming class data to instances. The inference theory of OWL does not extend to this case; meanwhile, database researchers have created models attuned to integration requirements (e.g., [WR05]). Third, the elegant approach of mediating with a general purpose inference engine may not be feasible to deploy worldwide (due to skill requirements and support by tiny companies); also, many inference engines run one instance at a time. In contrast, CLIO rightly chose to produce code (e.g., SQL, XML-languages) that a database server can optimize, parallelize, and run. Thus, AI approaches still require SQL skills. Finally, staff trained in database technologies greatly outnumber those with AI skills.

5. Unifying Data Exchange Theory and Business Objects' Update

Update is rarely considered in data exchange research, for several reasons. Read seems like lower hanging fruit, with fewer semantic difficulties and commit problems. Besides, data owners are typically reluctant to let outsiders update their systems of record. Instead, they allow updates to populate a staging table or stream, which they then apply to the "real" data using handwritten code or ETL scripts.

Yet there is another setting where updates *must* be translated between schemas and percolate into the database. In multi-tier systems, upper tier applications issue updates against business objects, and the database makes them persistent. Today, the necessary mappings are accomplished largely through procedural code. As a result, one MITRE customer estimates a cost over the next decade of $10-20M to assimilate new message data into its business objects, database tier, and GUIs.

Perhaps data exchange and view update ideas can help? The two theories clearly overlap -- both attempt to create instance mappings from a source to a target that will satisfy all relevant constraints; updates give additional hints on how the change should be achieved. Data exchange can be seen as a bulk view update of an empty table. The problem of merging with existing data is like a mix of Update and Insert. But none of the theories applies perfectly.

View update theory has been studied extensively [Kell86, Shu00], with the aim of taking a user's update of the view and determining an appropriate update to base data that has the intended effect on the view. The SQL standard identifies certain solvable cases, and provides DDL constructs by which an administrator can disambiguate. Considerable attention has gone to which views are reasonable to update.

Since there are many possible updates that preserve the intent of the view update, theory has been developed to find minimal changes. (For example, for inserting a view tuple, one should not be allowed to delete the entire database and then insert tuples to make the new one appear). [BaSp81] ask "What complement remains constant?" and prove a variety of desirable properties. But to employ such a theory in practice, one needs to explain to administrators how to determine the best constant complement and explain to developers the consequences of that answer. Good luck!

The OO developers who create business objects take a different viewpoint, and we believe the database community has done too little to support them. They let developers implement update methods (as procedural code). It is the developer's problem to make sure that this code correctly changes the base data. They write methods from scratch, unfortunately with no automated help based on the existence of constraints. Still, their approach has several strengths.

- They normally understand the real world semantics, so the updates they issue usually avoid cases that make little business sense and incidentally plague data exchange theories (e.g., "Update EMPLOYEE table to increase Total Salary by 10%", or "Change the office for *some* Employee in Department 'xyz'").
- Aborting the update is often the most appropriate response to a constraint violation, especially if one does not trust the user issuing the update to choose an appropriate repair. For example, if a low level clerk tries to add a million dollar order for an unknown customer, rejecting the request makes good business sense and of course preserves database integrity.

- Developers may write multiple methods that update the data, for different user communities. That is, one may not ask "what are THE update semantics for this view?"

Several system architects have told us that developers dislike SQL's way of letting a DBA customize view update semantics. Often the skills and knowledge reside with OO application developers, not the DBA. For different situations, they may want different update semantics (e.g., Abort versus run an interactive script [Kell86] versus fix automatically), which they achieve by writing several procedural methods.

While view update researchers (e.g., [Shu01]) have used constraint models, most attention to them today seems to be for data exchange. But data exchange research rarely considers "Abort" or isolating the tuples to be excluded from the final step.

Different constraint types would require different repair treatments.

- For *value constraints*, one must exclude illegal data or relax the constraint. For example, when supporting police investigations, one wants to declare what real world data is correct, but also allow deviations. It may be time to resume research on semantics of "soft" constraints.
- For *key constraints*, one must pick a winner or relax the constraint; in some situations, the most recent is the "winner".
- *Null not allowed* constraints seem particularly onerous -- we do not consider a Skolem constant an improvement over a null. One designs manual data input processes to capture all mandatory attributes; as we increasingly import data from outside, we have no such controls. Over the long term, we may wish to design applications and storage system (and their use of primary keys) to be more tolerant of missing values.
- *Foreign keys* resemble null not allowed, except that one might automatically repair the problem. But automated repair is not always appropriate – again, consider the example of the company not wanting to insert a huge order from a previously unknown customer by automatically generating a customer number.

Finally, access controls may complicate constraint enforcement. SQL delegation semantics removes part of the problem by insisting that update authority includes the right to check all constraints. Some business process designers may consider this inappropriate. In any case, one may not have the privileges to repair a problem on another table, or even to delete an existing record with the same key.

6. Conclusions

Pragmatic issues of concern to data exchange theorists and system builders have been discussed, including: (1) the need to decompose problems into automatable chunks plus residue problems that can be understood by ordinary developers and by nonprogrammers who supply domain knowledge; (2) the advantages of emphasizing automation of downstream more than upstream processes; (3) the strengths and weaknesses of database and AI approaches to data exchange, and the need for greater dialog between these communities; and (4) the need to address *update* in data exchange, especially to support n-tier systems.

Our goal has been to stimulate discussion for the benefit of both theory and system researchers, by discussing selected aspects of previous work. We have made broad statements to provoke thought and solicit feedback to correct any consequent errors or miscommunications.

Good computer science research is motivated by practical (or at least potentially practical) problems. While the best practitioners attempt to take relevant research into account, our focus here is to address the other side of the partnership – to highlight the ways in which skilled researchers can make their results more useful and widely accessible.

There are several benefits. First, it is in theorists' interests to help – research funding is influenced by perceived technology transfer, and computer science has been suffering. Second, theoreticians will undoubtedly discover interesting new insights and challenges from learning more about practitioners' concerns. Finally, one might even make a semi-serious data exchange argument.

The translation between formalists' language and that of system-oriented researchers who work with practitioners can itself be viewed as a data exchange problem. Who is better prepared to accomplish this mapping than a data exchange theorist? Here, too, it is important to examine which formalisms facilitate exchange and to try to tweak problems and theories to leave a manageable residue.

While some might argue that technology transition is not the theorists' problem, we believe that each theory community should develop two sorts of products: first for their own needs, and second for consumer communities.

References

[ArKn93] Y. Arens and C. Knoblock, "SIMS: Retrieving and integrating information from multiple sources," *ACM SIGMOD*, 1993

[BaSp81] F. Bancilhon, N. Spyratos: "Update Semantics of elational Views," *ACM Trans. Database Syst.* 6(4): 557-575, 1981

[Bert05] Bertossi et. al. Exchange, Integration, and Consistency of Data. Report on the ARISE/NISR Workshop, *SIGMOD Record* 34(3): 87-90, 2005

[CHS91] C. Collet, M. Huhns, W. Shen, "Resource Integration Using a Large Knowledge Base in Carnot," *IEEE Computer*, 24(12), December 1991

[DHN04] A. Doan, A.Y. Halevy, N. F. Noy, "Semantic integration workshop at the 2nd international semantic web conf (ISWC-2003)," *SIGMOD Record*, 2004

[GBMS99] C. H. Goh, S. Bressan, S. Madnick, M. Siegel: Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. *ACM Trans. Inf. Systems*, 17(3), 1999

[HEDI03] A. Halevy, O. Etzioni, A. Doan, Z. Ives, J. Madhavan, L. McDowell and I. Tatarinov, "Crossing the Structure Chasm," *Proceedings of the First Conference on Innovative Data Systems Research*, CIDR-2003

[HMH01] M. Hernández, R. J. Miller, L. Haas, "Clio: A Semi-Automatic Tool For Schema Mapping," System Demo, *Proceedings of ACM SIGMOD*, 2001

[IMHB04] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga, "CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies," *Proceedings of ACM SIGMOD*, Paris, France, 2004

[Kell86] Keller, Arthur. Choosing a View Update Translator by Dialog at View Definition Time. *Very Large Data Base Conference*, Kyoto, Japan (1986)

[Ko05] P. Kolaitis, "Schema Mappings, Data Exchange, and Metadata Management" PODS 2005 Invited Talk

[MRB03] Melnik, S., E. Rahm, P. A. Bernstein, "Rondo: A Programming Platform for Generic Model Management," *Proc.ACM SIGMOD* 2003, pp. 193-204

[MRS+05] P. Mork, A. Rosenthal, L. Seligman, J. Korb, K. Samuel, "Integration Workbench: Integrating Schema Integration Tools," submitted to *International Workshop on Database Interoperability at ICDE*, 2006

[MS89] V. Markowitz, A. Shoshani, "Abbreviated Query Interpretation in Extended Entity-Relationship Oriented Databases", *Entity Relationship Conference*, 1989.

[RaBe01] Rahm, E. and Bernstein, P. 2001. On matching schemas automatically. *VLDB Journal* 10, 4.

[RoSe94] A. Rosenthal, L. Seligman, "Data Integration in the Large: The Challenge of Reuse", *Conf. on Very Large Data Bases*, Santiago Chile, Sept. 1994.

[RSRM01] A. Rosenthal, L. Seligman, S. Renner, F. Manola, "Data Integration Needs an Industrial Revolution," *International Workshop on Foundations of Models for Information Integration (FMII-2001)*, Viterbo, Italy, September 2001

[Shu00] Hua Shu, Using Constraint Satisfaction for View Update, *Journal of Intelligent Information Systems* Volume 15, Number 2 September 2000 Pages: 147 - 173

[SE05] P. Shvaiko, J. Euzenat "A Survey of Schema-based Matching Approaches", *Journal on Data Semantics* LNCS 3730, 2005

[W3C04] World Wide Web Consortium, *Web Ontology Language Overview*, http://www.w3.org/TR/owl-features/, February 2004

[WyRo05] C. Wyss, E. Robertson, Relational Languages for Metadata Integration, *ACM TODS*, June 2005.

Acknowledgments

The authors thank Barbara Blaustein for her very helpful comments. In addition, we benefitted from discussions with Murali Mani and Peter Mork and from comments of the anonymous reviewers.