

Tool Requirements for Precise Data Integration

Philip A. Bernstein

Microsoft Research
philbe@microsoft.com

Abstract

We need to help data architects solve precise data integration problems by specifying non-procedural mappings. This requires a software framework that offers generic components that are reusable in different kinds of mapping tools. It also requires solutions for schema evolution, to revise non-procedural mappings in response to changes in the interfaces they depend on.

1. Two Kinds of Information Integration

We begin by scoping the problem domain. There are two main usage scenarios for information integration:

- information discovery – gathering the most relevant information in response to an exploratory request.
- precise data integration – mapping data between a target schema and one or more data sources to enable queries and updates over the target schema or translation of data from the sources to the target.

Information discovery is closely related to information retrieval. It involves identifying information that helps an end user. Usually, there is no well-defined notion of correct answer. The answer is a best-effort. Examples are web search and (the vision of) the semantic web. The main relevant technologies for information discovery are information retrieval, machine learning, knowledge representation and reasoning, and natural language processing.

Precise data integration is needed when there is a well-defined notion of correctness of the integration result. Therefore, the mapping between sources and target must be in a formal language with precise semantics. Examples are message translation for E-commerce, query mediation over heterogeneous databases, data warehouse loading, database wrapper generation, data exchange, and database-driven portals. The main relevant technologies for precise data integration are schema management, mapping management, and query processing. The rest of this paper focuses on precise data integration.

2. The Need for Tools

Anecdotal evidence suggests that 40-50% of the work in IT departments of large enterprises is that of precise data integration. Many, perhaps most, of the information integration problems faced in science, engineering and medicine are also those of precise data integration.

Performing precise data integration requires a deep understanding of the meaning of the data to be integrated. Today, at best there is a weak formal specification of that meaning expressed in a schema definition language. It is weak in that it says little (more often nothing) about integrity constraints, data quality, intended usage, data lineage, etc. Given that the specification of meaning is weak and that the result of integration must be precise, it is currently hopeless to fully automate the integration process. A human data architect must be in the loop. The best we can hope for now, and probably for a long time to come, is to offer tools that reduce the time and level of technical expertise that the data architect requires.

We should be careful to avoid being overly ambitious when defining the goals of such tools. No matter how good the tools, for non-technical reasons a data architect usually needs weeks or months to solve a precise data integration problem. It takes time to read documentation, talk to users about requirements, compare data instances to their claimed semantics, arrange to gain access to the data, test an integration solution and have it be evaluated by end users, scale up a tested solution, and change organization processes related to acquiring data, altering its format and meaning, and changing how it is used. Tools can do little to reduce the elapsed time for these activities.

3. How to Develop Tools

Commercial products are available for many integration problems. Here are some examples of kinds of tools:

- Extract Transform and Load tools, to simplify the programming of scripts to extract data from sources, clean it, reshape it, and load it into a data warehouse.
- Message mapping tools, to simplify the programming of message translation between different formats

- Graphical query design tools, to define a mapping between source and target schemas.
- Portal design tools, to map data sources to controls that can be conveniently displayed.
- Wrapper generation tools, for example, to produce an object-oriented wrapper for a relational database.

These tools are useful, but they can and should be much better. Here are some opportunities for improvement:

- Better mapping languages are needed. Mappings are often either highly procedural and hence difficult to write and understand, or are expressed abstractly in a graphical language whose semantics is vague (e.g., as lines connecting schema elements). When a precise non-procedural mapping language is offered, it usually has limited expressiveness.
- Large schemas are hard to visualize. Mappings over large schemas are very hard to define and understand. Better visualization techniques are badly needed. Moreover, better techniques are needed for modularizing schemas and mappings so they need not all be displayed at once.
- There is little help in automating the definition of mappings except in obvious cases. Although there has been a lot of research on schema matching, we know of no end-user studies demonstrating how useful it is in practice.
- There is usually no help offered by tools to validate mappings relative to constraints on the source and target schemas.
- Solutions to design problems must round-trip. That is, if a developer modifies an artefact generated by an integration tool (e.g., a database schema or mapping), the tool must be able to reverse engineer those modifications back to the design description.
- There is usually no help in coping with the evolution of schemas, which requires the modification of mappings that reference the changed schemas.

Each of the above problems is quite difficult. The market for integration tools is small, relative to the size of the market for end-user software. Therefore, the number of engineers that a tool vendor can afford to assign to each problem is small. To address this problem, we need to develop a software framework that offers generic solutions to these problems that are reusable in different kinds of tools. So far the results in this direction are mixed.

The good news: There has been some agreement on the need for certain mapping manipulation operations, such as schema matching, mapping composition, and code generation from a non-procedural mapping specification. Algorithms for these operations have been developed, some of which have been applied to practical problems.

The bad news: There has been little concrete evidence that the implementation of these operations is reusable across usage scenarios. Not many such implementations have been reported in the literature, so it's too early to draw firm conclusions. But so far, practical implementations that use such operations appear to be quite sensitive to the choice of metamodels, mapping language, and other scenario-specific requirements. One exception is schema matching. Useful solutions have been developed that use only a modest amount of schema information (e.g., element names, element data types, and schema structure). They should therefore be portable across metamodels, but as far as we know, no success stories have been reported.

I am still cautiously optimistic that reusable implementations of generic operations can be developed. However, the road ahead appears to be longer and more circuitous than I originally had hoped. It may be that we need many more custom solutions to real world usage scenarios to be able to identify complete requirements that generic and hence reusable implementations of operations require.

4. Schema Evolution

Suppose we make huge progress on the above problems. Imagine a world where most data integration problems are solved by having a data architect define non-procedural mappings annotated with custom code where necessary. Suppose tools are available to help define, validate, and test such mappings. In that world, the next big problem that will be faced by data architects is schema evolution.

Mappings are defined over interfaces that change regularly. New releases of applications have new interfaces and updated database and message schemas. User requirements for accessing data change. Applications depend on interfaces to web services that they use. Mappings defined on these changing interfaces must be revised.

The trend is for users to buy applications rather than build them. Application products need to be customized by value-added resellers (VARs) who repackage an application for a vertical market, and then by users who have enterprise-specific requirements. Each new release of an application aims to support all old published interfaces, to avoid breaking customizations. But this is hard to do and is rarely a complete solution. For example, customizations include extensions to forms and databases, which can conflict with the new release. New releases may exploit new platform capabilities (e.g., new database and GUI features), which force changes to old interfaces.

There is a big literature on software maintenance and on schema evolution in particular. Database views offer some help. But beyond that, there are hardly any products that address schema evolution. Why is that and what can we do to improve the situation?