# Mappings as Building Blocks for Complex Integration

Lucian Popa
IBM Almaden Research Center

## 1 Introduction

Mappings between different representations of data are a common and important component of many information integration tasks. A schema mapping is a useful abstraction that specifies how the data from a source format can be transformed into a target format. We have developed tools and algorithms for generating such mappings between schemas and also methods for generating executable code that transforms data according to the specification given by a mapping [HHH$^+$05]. We have also developed, to a large extent, the necessary theory needed to articulate the semantics of mappings and of their operations (e.g., composition of two consecutive mappings) [Kol05].

Often times, a mapping (or the transformation implied by it) is only a part of a larger set of components (e.g., other mappings, transformations, or black-box procedures) that need to be orchestrated together in what can be called an "integration application" or "integration job". Examples of systems where you can program such applications are ETL engines or data mashup environments. However, the level of abstraction in these systems is low, there is little automation, and the resulting integration application is hard to optimize, manage and reuse.

To make integration accessible on a much larger scale, we need to provide the various users and application programmers with the ability to *easily combine existing components (e.g., mappings)*, hopefully with as much automation as possible, and build the integration code they need. Some of the key challenges in building such an architecture are: (1) defining clear interfaces and "standard" languages for components, (2) defining and implementing languages for scripting applications based on these components, (3) developing repositories for storing the needed metadata (schemas, mappings, constraints, queries, user policies and requirements, etc.), and (4) automating the "assembly" of the integration task from the individual components (instead of having a human expert programming the integration task).

Some of these challenges are already addressed, to some extent, by existing research. For example, model management [Ber03] is an integrating framework that can express at a high level of abstraction various data management tasks in terms of operations on schemas and mappings. However, more is needed to bring the high level of abstraction "down" to the concrete run-time (execution engine, access constraints, performance considerations, etc.) and also to the concrete specifics of an application (user rules and policies, etc.) [Haa07]. Also, more is needed to further ease the programming process (even when a high level of abstraction such as model management is used).

One could argue that challenge (4) listed above is still far beyond our current capabilities today. Yet, there are several directions that can be pursued to make integration more automatic. One particularly important aspect here is in fact related to challenge (3) above. More concretely, a *metadata repository* can be used to *share and reuse* artifacts (e.g., mappings) accross multiple integration applications in the same domain. The key assumption, intuitively, is that there are only finitely many ways in which a piece of data can be converted from a certain type to another type. Moreover, applications in the same domain tend to use the same "common" types of data (even though the concrete representations may vary). For example, most HR applications will manipulate data about employees, jobs, departments, etc. Thus, there is a relatively small set of common and frequently used types that are "buried" in a potentially large number of heterogeneous schemas and data sources. Hence, there is significant potential for reuse.

The mappings that are stored in a repository could be useful for a new application even when they do not match exactly the needs of the new application (because their schemas may not match exactly, or the "desired" semantics or performance may not match exactly). Often times the architect of the new integration application may be able to use such existing mappings as a starting point of the design. Furthermore, with the help of mapping tools, existing mappings can be semi-automatically "adapted" or "deployed" in a new (but similar) context.

There are several interesting and quite fundamental research questions that arise in the context of using a metadata repository for schemas and mappings. Here is an incomplete list: (1) indexing the repository based on "types" (rather than schemas), (2) standardizing mappings between "types" (rather than schemas), (3) finding "similar" schemas in the repository, (4) finding "similar" mappings in the repository, (5) processing, reasoning about and comparing different mapping alternatives, (6) methods for visualization or browsing of the repository, etc. An additional issue is being able to represent black-box procedures that contained customized code. Although such procedures could be represented, in a simplified way, as uninterpreted mappings from their input type to their output type, a key issue here is providing enough metadata to be able to distinguish and choose among different such procedures.

To give a very simple example of how the repository could be used, assume that in the new application we need a mapping from a schema $\mathbf{S}$ to a schema $\mathbf{T}$. The output of a query on the metadata repository could return a set of potential "paths" of mappings between two schemas $\mathbf{S}'$ and $\mathbf{T}'$ that are most "similar" to $\mathbf{S}$ and $\mathbf{T}$, respectively. The next steps will be then to filter, compose, merge or simply union these mappings to assemble an end-to-end mapping from $\mathbf{S}'$ to $\mathbf{T}'$. The last step (perhaps interleaved with the previous steps) would be to further solve the differences between $\mathbf{S}$ and $\mathbf{S}'$ (and $\mathbf{T}$ and $\mathbf{T}'$) to obtain an end-to-end mapping from $\mathbf{S}$ to $\mathbf{T}$.

As the example suggests, there is a small set of operations on mappings that is particularly useful for the "assembly" process. Some operations are relatively simple: filtering and union, for example.[1] Other operations on mappings are more complex: merging of consecutive mappings (i.e., composition), merging of parallel mappings, inverting mappings, comparison of different mapping paths, visualization of the differences between mapping alternatives. Some of these operations have already been addressed to some extent (composition and inversion, for example) but some are still largely unexplored.

**Acknowledgements** This short text is a partial reflection of several discussions with Michael Gubanov, Laura Haas, Mauricio Hernández, Howard Ho, Phokion Kolaitis, Renée Miller, Hamid Pirahesh, Ioana Roxana Stanoi, Wang-Chiew Tan and others.

## References

[Ber03] P. A. Bernstein. Applying Model Management to Classical Meta-Data Problems. In *Conference on Innovative Data Systems Research (CIDR)*, pages 209–220, 2003.

[Haa07] Laura M. Haas. Beauty and the beast: The theory and practice of information integration. In *ICDT*, pages 28–43, 2007.

[HHH$^+$05] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *SIGMOD*, pages 805–810, 2005.

[Kol05] P. G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 61–75, 2005.

[1] A union of mappings, however, will often require correlation of their results (e.g., merging of duplicate objects, grouping, etc.).