Gain Control over your Integration Evaluations

Patricia C. Arocena
University of Toronto
prg@cs.toronto.edu

Radu Ciucanu
University of Oxford
radu.ciucanu@cs.ox.ac.uk

Boris Glavic
Illinois Inst. of Technology
bglavic@iit.edu

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

ABSTRACT

Integration systems are typically evaluated using a few real-world scenarios (e.g., bibliographical or biological datasets) or using synthetic scenarios (e.g., based on star-schemas or other patterns for schemas and constraints). Reusing such evaluations is a cumbersome task because their focus is usually limited to showcasing a specific feature of an approach. This makes it difficult to compare integration solutions, understand their generality, and understand their performance for different application scenarios. Based on this observation, we demonstrate some of the requirements for developing integration benchmarks. We argue that the major abstractions used for integration problems have converged in the last decade which enables the application of robust empirical methods to integration problems (from schema evolution, to data exchange, to answering queries using views and many more). Specifically, we demonstrate that schema mappings are the main abstraction that now drives most integration solutions and show how a metadata generator can be used to create more credible evaluations of the performance and scalability of data integration systems. We will use the demonstration to evangelize for more robust, shared empirical evaluations of data integration systems.

1. INTRODUCTION

Traditional evaluation of integration systems is generally limited to collections of real-world datasets that are shared by the community (e.g., the Amalgam Schema and Data Integration Test Suite[1], the Illinois Semantic Integration Archive[2], or the Thalia testbed[3]), or to synthetic scenarios based on standard patterns (e.g., star schemas). The scope of such evaluations is usually restricted to a specific feature of an approach, and consequently, they are often difficult to reuse and compare.

[1] http://dblab.cs.toronto.edu/~miller/amalgam/
[2] http://pages.cs.wisc.edu/~anhai/wisc-si-archive/
[3] http://cise.ufl.edu/research/dbintegrate/thalia/

Proceedings of the VLDB Endowment, Vol. 8, No. 12

Complicating matters, the types of input required and output produced by different integration solutions has historically been quite varied. A matching system may take schemas as input and output sets of pairs of attributes, a schema mapping system may take the same input but output a schema mapping or a transformation program, and a federated data integration system may take the schemas, a schema mapping and set of (target) queries as input and output a set of rewritten (source) queries. Community shared integration scenarios typically provide a *gold-standard* or correct output for at best a single task limiting their usefulness. The same could be said of (the much more extensive) shared repositories of data for machine learning, like the UCI Machine Learning (ML) Repository[4] that provide inputs for different learning tasks (clustering, classification, regression, etc.). But unlike ML, the integration community has not agreed upon a set of quality or performance metrics to judge our results. How close is a mapping to a gold standard mapping? How should we compare the performance of query rewriting systems - by the speed of rewriting or by the speed of the rewritten queries?

Database benchmarks such as TPC (including TPC-DI[5], a Data Integration Benchmark) and XMark[6] rely on fixed schemas. For most data integration tasks, these fixed-schema benchmarks are not suitable. Integration systems must be tested on schemas of different sizes and characteristics to understand their generality and to quantify their performance (accuracy and speed).

In this demonstration, we motivate the need for a metadata generator. As with the data generators that accompany fixed-schema benchmarks, a metadata generator must be able to control the size and characteristics of the metadata it generates. In addition, the generator must be able to generate metadata that meets the needs of a variety of integration tasks. In the next sections, we overview the requirements of common integration tasks and present a set of metadata generator requirements. We then discuss our generator implementation and demonstration scenarios.

2. INTEGRATION TASKS

We review some common integration tasks and their evaluation requirements. Each of these tasks requires a set of input elements (e.g., schema(s), constraints, or attribute correspondences) and produces a set of output elements (e.g.,

[4] http://archive.ics.uci.edu/ml/datasets.html
[5] http://www.tpc.org/tpcdi/
[6] http://www.xml-benchmark.org/

a database instance, correspondences, or mappings). Common across these tasks, is that to evaluate the performance, accuracy or completeness of a system implementing a task, we need a metadata generator that produces the required metadata inputs and allows a user to control properties of such inputs. Schema matching is perhaps the simplest of integration tasks. For matching, the input is a pair of schemas and the output is a set of attribute pairs (correspondences). To evaluate a schema matching system, a metadata generator must generate input schemas with different characteristics and naming conventions, and produce a gold-standard output (the correspondences). The quality of a schema matching is measured by comparing the generated correspondences against the ground-truth using statistical measures such as precision, recall, or F-measure [4].

2.1 Schema Mapping Generation

Systems that generate mappings between schemas take a pair of schemas, constraints, and correspondences as input, and produce a mapping. The performance of such a task is measured as the time needed to generate the mappings automatically or as the effort required by a human to produce the expected results. Quality measures for mappings are more diverse and complex than for schema matching as a mapping can be expressed in several equivalent ways. Using measures such as precision to compare a mapping to a ground-truth mapping is meaningless, because there are currently no normal forms defined for most mapping languages. Another drawback of measures such as precision and recall is that they only measure total matches. This type of 0-1 comparison between output and expected output is not very useful for mappings, because a mapping may only be partially correct. These issues have lead to the development of new quality measures which are evaluated over instance data [1].

The state-of-the-art in schema mapping evaluation is ST-Benchmark [2], a system that uses a set of primitives. Each primitive is a pair of a source schema, a target schema, along with a (gold-standard) transformation from the source to the target. Primitives can be combined to produce pairs of schemas of variable size that can be used as input to evaluate mapping systems. Each system can then be evaluated on how close it comes to producing the desired (gold-standard) mapping. Importantly, STBenchmark only produces transformations and does not permit developers to easily introduce new primitives. STBenchmark also does not permit direct control over the generation of schema constraints.

Input	Schemas, Contraints, Instance Data, Correspondences
Output	Mappings, Transformations

2.2 Data Exchange

In data exchange, an instance of a source schema is translated into an instance of a target schema. The input for data exchange is a pair of schemas, their constraints, optionally a source instance, and a mapping between the schemas. The output is a data transformation program that given an instance of the source schema, generates an instance of the target schema that satisfies the mapping and target constraints. When evaluating the quality of the generated transformations we face similar challenges as for mapping generation. We must be able to compare different transformation programs. A transformation program is correct if given a source instance $\mathcal{I}$ it produces a target instance $\mathcal{J}$ where $(\mathcal{I}, \mathcal{J})$ satisfy the mapping. Of course, mappings may permit many solutions. Some quality measures for data exchange are based on comparing the generated target instance against a ground truth.

Input	Schemas, Constraints, (Source Instance), Mappings
Output	Executable Transformations, (Target Instance)

2.3 Virtual Data Integration

Virtual data integration (or federation) rewrites queries posed on a global or mediated schema into queries on local schemas, using mappings between the global and local schemas. The input for a virtual data integration system includes the schemas, mappings, instances of the local schemas, and a set of queries. The expected output are either rewritten queries or the results of these queries. Performance of a virtual integration system can either be measured as the time it takes the system to generate the rewritten queries or the time it takes to execute these queries. The quality of the result can either be evaluated as the difference between the generated query result and expected results or by directly comparing the expected and produced queries.

The state-of-the-art in evaluating virtual data integration is to use *ad hoc* mapping generators that create mappings having a specific structure (e.g., chain joins or star joins).

Input	Schemas, Instance Data, Mappings, Queries
Output	Rewritten Queries, Query Results

2.4 Mapping Operators

In addition to systems that generate mappings, there is a large body of work on manipulating sets of mappings through operators. Typical mapping operators include composition, inversion, normalization, mapping evolution or adaptation, and correlation of mappings [1]. The input for these operators is a set of mappings, and for some operators the schemas over which the mappings are defined. Mapping operators such as composition require that the input mappings are correlated in a certain way. An obvious performance measure for mapping operators is execution time. The quality of an operator's output can be measured by comparing the generated mappings against the ground truth using one of the methods introduced above for mapping generation.

The state-of-the-art in evaluation of mapping operators include the use of a suite of schema evolution primitives [5] (similar, but not identical, to the primitives of STBenchmark). A different set of primitives were used to evaluate mapping adaptation [8]. Notably these primitives allow the creation of synthetic mappings (ST-tgds) of different sizes and characteristics, but they do not generate schema constraints or other types of metadata.

Input	Schemas, (Instance Data), Mappings
Output	Mappings

Additional tasks such as peer-to-peer (P2P) data integration and schema evolution can likewise be formalized.

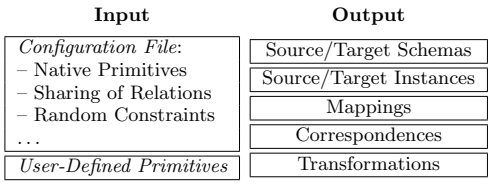

Figure 1: Metadata Generator Input and Output

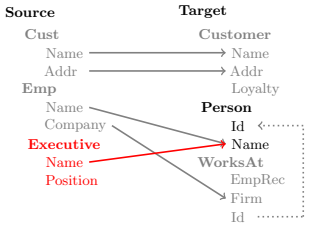

Figure 2: Multiple Primitives with Target Sharing

3. GENERATOR REQUIREMENTS

We now discuss requirements for a generic metadata generator that would ease the empirical evaluation of data integration systems implementing typical integration tasks.

R1: Generation of Different Types of Metadata. We know integration tasks each take different types of inputs and produce different outputs. A metadata generator should be able to produce all these types of inputs, and also produce a "ground truth" output for evaluating quality measures.

R2: Concise Specification of Scenario Characteristics. Empirical evaluations require a metadata generator that can produce integration scenarios with controlled characteristics. The user should be able to provide such a specification. For example, to test how well a system scales in the instance size, a researcher would use the metadata generator to generate a set of scenarios with the same schema and mapping characteristics, but different instance sizes. Similarly, to measure how sensitive the quality of transformations produced by a data exchange system is to correlations among elements in the source and target schema, a researcher could use the generator to produce a set of schemas with different amounts of correlation. The generator should support creation of a set of scenarios where some characteristics are fixed and some characteristics are varied.

R3: Realistic and Diverse Synthetic Scenarios. Synthetic scenarios produced by a metadata generator should be similar in structure and content to real-world scenarios. In particular, the mappings and schemas should follow patterns that are common in real-world scenarios. Instance data values should be drawn from realistic distributions.

R4: Scale Real-world Scenarios and Data Sets. Real-world integration scenarios and datasets are great tools for evaluating integration systems. Because these scenarios are usually small, a metadata generator should be able to scale them while preserving their characteristics. In addition, we would like to combine scaled real-world and synthetic scenarios to create scenarios with fine-tuned characteristics.

R5: Quality Measure Library. Some measures used to evaluate integration tasks are quite complex. A metadata generator should ship with a library of quality measure implementations to lower the bar for wide adoption.

4. SYSTEM OVERVIEW

We use iBench[7], a highly-configurable and flexible metadata generator that fulfills the requirements presented in Section 3, to demonstrate how to generate metadata for evaluating integration tasks. The input of iBench is a configuration file where the user specifies desired scenario characteristics. iBench can generate source and target schemas, instances, mappings and transformations (see Figure 1), and build complex scenarios from typical mapping primitives which act as "templates" for defining metadata fragments. The system currently supports ~15 *native primitives*, which correspond to typical patterns such as adding or deleting attributes from a relation, vertically or horizontally partitioning a relation, etc. In contrast to STBenchmark, iBench supports sharing of relations across primitives, generation of additional types of metadata, and user-defined primitives. In the configuration file, the user can specify existing scenarios to be loaded as user-defined primitives (UDPs). Using UDPs we can scale real-world scenarios and combine them with the native primitives to produce complex (and realistic) scenarios.

EXAMPLE 4.1. The user requests a scenario with three mapping primitives: copying a relation and adding a new attribute, vertically partitioning a relation (VP), and copy a relation while both adding and deleting attributes. The user also requests sharing of relations among primitives. A possible output of iBench is shown in Figure 2. The source relation *Cust* is copied to a target relation *Customer* where we add an attribute *Loyalty* and the source relation *Emp* is vertically partitioned into target relations *Person* and *WorksAt*. The last primitive (in red) shares its target relation with the VP primitive. We depict the attribute correspondences by solid lines and the foreign key (FK) constraints by dotted lines. □

By default iBench uses a custom XML format for storing a generated integration scenario. We can easily extend it to produce the output in a task-specific format, as we have already done for the evaluation of MapMerge [1]. iBench comes bundled with a data generator based on ToXgene[8] that is used to create source/target instance data.

5. DEMONSTRATION OVERVIEW

The demo scenario consists of four parts. First, we show how parameters in the generator's input configuration file can be set to fit a specific integration task and control the characteristics of the generated metadata. Second, we illustrate how UDPs can be used to create and enrich synthetic scenarios. The first two scenarios will make the attendees aware of the kind of flexibility we need to gain control over our integration evaluations. Third, we show a complete workflow for evaluating MapMerge [1] using generated synthetic metadata and instance data that scales. Fourth, we present new empirical insights about MapMerge that were enabled by our new evaluation harness. The last two scenarios will make the attendees aware of the importance of comparing integration solutions on a level playing field.

[7] `http://dblab.cs.toronto.edu/project/iBench/`

[8] `http://www.cs.toronto.edu/tox/toxgene/`

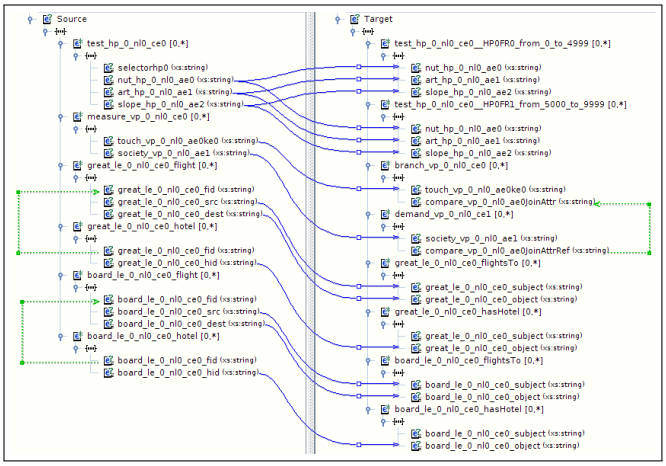

Figure 3: Visualization of Synthetic Schemas and Correspondences using Clio

Controlling Metadata Generation

In the first part of the demo, the attendees will discover how we can use a highly-configurable metadata generator to fit specific integration tasks. First, we will use simple configurations to illustrate that primitive-based generation of metadata is an effective way of generating realistic and diverse synthetic integration scenarios. Starting from a simple configuration with a single primitive, we will incrementally activate additional primitives and show the impact on the generated output. We will showcase how to generate integration scenarios with different types of metadata, including schemas, correspondences, and much more (Requirement R1). We also will visualize the generated metadata in well-known research systems, including Clio (e.g., Figure 3) and ++Spicy [6, 7]. Participants will learn how to control the characteristics of the generated metadata by editing the generator's configuration file (Requirements R2 and R3).

Scaling Real-world Scenarios

Next, we will use the generator to scale real-world scenarios while maintaining their characteristics, and show how to combine real-world with synthetic scenarios (Requirement R4). The demonstration will use a biological scenario, a bibliography scenario based on Amalgam, and scenarios simulating graph databases.[9] Participants will learn how to use UDPs to scale an imported scenario in various dimensions (e.g., schema size) and combine it with synthetic metadata.

Evaluating Integration Systems

Attendees will be able to run a complete workflow for evaluating the MapMerge operator [1]. This workflow consists of three steps: (i) generating metadata and data using iBench, (ii) generating Clio, MapMerge, and ++Spicy mappings, and (iii) evaluating the performance of Clio, MapMerge, and ++Spicy via some measures (Requirement R5). An important measure is the size of the target instance. Intuitively, among schema mappings that correctly translate source data, we would prefer mappings that produce smaller target instances because this means that they produce less

[9]These scenarios are available from our iBench webpage: `http://dblab.cs.toronto.edu/project/iBench/`

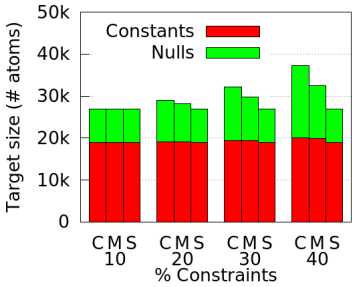

Figure 4: Clio/MapMerge/++Spicy Comparison Plot

invented values [3] and consequently better handle the data incompleteness. Using iBench, the attendees will produce diverse scenarios (Requirements R3 and R4) and observe the impact of correlating independent schema mappings. We will automatically generate Gnuplot scripts to plot the results (e.g., as shown in Figure 4, where "C" stands for Clio, "M" for MapMerge, and "S" for ++Spicy).

Sharing Evaluation Insights

Last we will share some of the insights we have gained through performing evaluations with iBench. For instance, we will highlight some cases where MapMerge performs very well, which were not considered in the initial evaluation of this approach [1]. This last part of the demonstration scenario will motivate the need for stress testing integration solutions by going beyond their existing evaluations. Our ultimate goal is to convince attendees of the value of using a metadata generator such as iBench in their empirical evaluations.

6. REFERENCES

[1] B. Alexe, M. A. Hernández, L. Popa, and W. C. Tan. MapMerge: Correlating Independent Schema Mappings. *VLDB J.*, 21(2):191–211, 2012.

[2] B. Alexe, W. C. Tan, and Y. Velegrakis. STBenchmark: Towards a Benchmark for Mapping Systems. *PVLDB*, 1(1):230–244, 2008.

[3] P. C. Arocena, B. Glavic, and R. J. Miller. Value Invention in Data Exchange. In *SIGMOD*, pages 157–168, 2013.

[4] Z. Bellahsene, A. Bonifati, F. Duchateau, and Y. Velegrakis. On Evaluating Schema Matching and Mapping. In *Schema Matching and Mapping*, pages 253–291. Springer, 2011.

[5] P. A. Bernstein, T. J. Green, S. Melnik, and A. Nash. Implementing Mapping Composition. *VLDB J.*, 17(2):333–353, 2008.

[6] R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema Mapping Creation and Data Exchange. In *Conceptual Modeling: Foundations and Applications*, pages 198–236, 2009.

[7] B. Marnette, G. Mecca, P. Papotti, S. Raunich, and D. Santoro. ++Spicy: an OpenSource Tool for Second-Generation Schema Mapping and Data Exchange. *PVLDB*, 4(12):1438–1441, 2011.

[8] C. Yu and L. Popa. Semantic Adaptation of Schema Mappings when Schemas Evolve. *VLDB*, pages 1006–1017, 2005.