IQ-METER – An Evaluation Tool for Data-Transformation Systems

Giansalvatore Mecca [1], Paolo Papotti [2], Donatello Santoro [1,3]

[1] *Università della Basilicata – Potenza, Italy* giansalvatore.mecca@gmail.com
[2] *QCRI – Doha, Qatar* ppapotti@qf.org.qa
[3] *Università Roma Tre – Roma, Italy* donatello.santoro@gmail.com

***Abstract*—We call a data-transformation system any system that maps, translates and exchanges data across different representations. Nowadays, data architects are faced with a large variety of transformation tasks, and there is huge number of different approaches and systems that were conceived to solve them. As a consequence, it is very important to be able to evaluate such alternative solutions, in order to pick up the right ones for the problem at hand. To do this, we introduce IQ-METER, the first comprehensive tool for the evaluation of data-transformation systems. IQ-METER can be used to benchmark, test, and even learn the best usage of data-transformation tools. It builds on a number of novel algorithms to measure the quality of outputs and the human effort required by a given system, and ultimately measures "how much intelligence" the system brings to the solution of a data-translation task.**

I. INTRODUCTION

The problem of translating data among heterogeneous representations is a long standing issue in the IT industry and in database research. The first data translation systems date back to the seventies. In these years, many different proposals have emerged to alleviate the burden of manually expressing complex transformations among different repositories, so that today we have a very broad class of systems conceived for a variety of purposes, including schema-matching, schema-mappings and data-exchange, data-integration, ETL (Extract-Transform-Load), object-relational mapping, data-fusion, and data-cleaning, to name a few.

The number of approaches is justified by the ample variety of scenarios that data architects need to deal with. For some of them the declarative techniques proposed in the literature are rather effective; others are such that only procedural and more expressive systems, like those used in ETL, are of real help. Therefore, a crucial design decision that may strongly impact the success of a project is to pick up the right tool for a given translation task.

To answer this question, we must be able to compare and classify systems coming from different inspirations and different application domains. Surprisingly enough, very little work has been conducted in this direction. In fact, only a few early benchmarks exist [1], [2], [3], and they are rather limited in scope, both in terms of the class of systems they consider, and of the aspects they evaluate.

IQ-METER is the first comprehensive tool for the evaluation of data-transformation systems conceived to address this concern. It can facilitate data architects' work on complex scenarios with both commercial and research systems. It is based on three core ideas introduced in [4]:

(i) a definition of a *data-transformation* system that is sufficiently general to capture a wide class of tools – including research schema-mapping systems, commercial mapping systems and transformation-editors, and ETL tools – and at the same time tight enough for the purpose of our evaluation;

(ii) a new fast algorithm to measure of the *quality* of the outputs produced by a data translation tool on a given mapping scenario;

(iii) a natural definition and measure of the *user-effort* needed to achieve such quality.

Ultimately, these techniques allow us to measure the *level of intelligence* of a data-transformation tool, defined as the ratio between the *quality* of the outputs generated by the system on a given task, and the *amount of user effort* required to generate them. The lower the effort required to obtain results of the best quality with a system, the higher its IQ. IQ-METER builds on these techniques, and provides sophisticated functionalities for the evaluation of data transformation tools.

IQ-METER in Action IQ-METER is used to run *evaluation sessions* on data-transformation systems (DTSs). In our view, a DTS is any tool capable of executing *transformation scenarios* (also called mapping scenarios or translation tasks). An evaluation session may consider a single DTS, or several DTS that need to be compared to each other.

Regardless of the way in which transformations are expressed, in our setting transformation scenarios require to translate instances of one or more source schemas into instances of a target schema. The transformation system is seen as a black box, of which we are only interested in the input-output behavior. For each transformation task, inputs to the system are: (a) the source schemas; (b) the target schema; (c) the source instances that need to be translated.

We do not constrain the way in which the translation is expressed by the DTS, as long as it is input by the user through a graphical user-interface and then stored on the disk as a *scenario file* using some readable format (typically, systems store the transformation as an XML file). To handle a DTS, it needs a *driver*, i.e., a module capable of parsing scenario files generated by a DTS. A snapshot of the system in action on three different systems – ++SPICY, Altova MapForce, and CloverETL, is shown in Figure 1.

Given a scenario, an IQ-METER evaluation session must also specify an *expected output*, i.e., a target instance that is considered as the right output for the given transformation, as

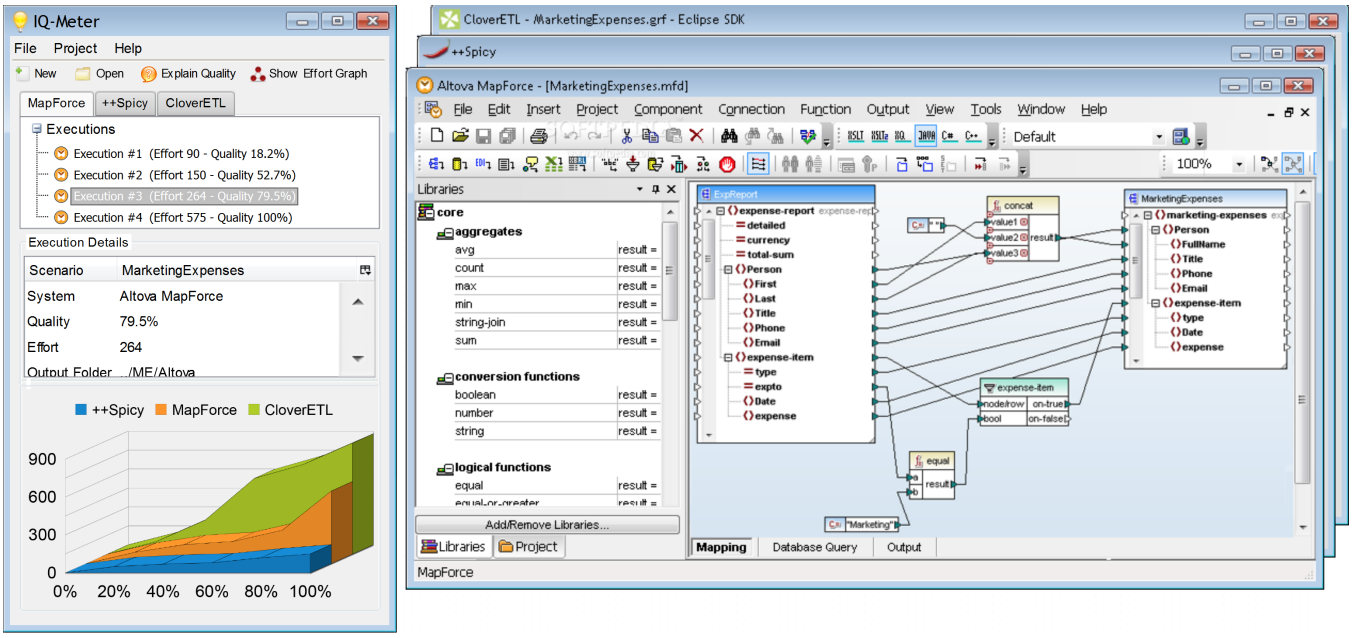

Fig. 1: IQ-METER in action

it will be discussed in more detail in the next section. The session consists of a sequence of executions of the various DTSs, in which the user progressively refines a specification of the intended transformation using the DTS GUI, and then runs the DTS engine to actually translate the data. IQ-METER measures the quality of the outputs and the amount of effort for the different executions using the algorithms introduced in [4], and dynamically shows a detailed evaluation report about the session. The most prominent feature of this report is a *quality-effort* graph, that shows how the quality achieved by a DTS varies with different specifications of the mapping, and therefore with different levels of efforts, as shown in Figure 1. Intuitively, the smaller is the area below the plot for a given system, the lesser is the effort required to achieve high quality outputs. Higher quality with lesser effort means higher effectiveness for the given task, and ultimately "more intelligence".

We introduce the main techniques behind the system in the next sections. In the meanwhile, we want to emphasize that IQ-METER is a unique tool for evaluating data transformation systems. In its most basic form, it can be used as a test tool for new DTSs, in order to measure the correctness of the transformations expressible with the system. However, it has a number of other interesting applications, as follows:

(a) when used on existing benchmarks, like [2], [3], it brings two significant advantages; on the one side, it is general and extensible, since it allows to mix and compare systems with different inspirations, like, for example, schema-mapping tools and ETL tools; in addition, previous benchmarks only considered the following typical question: "is this tool capable of expressing this scenario?" on the contrary, IQ-METER provides deeper insights about the effectiveness of a DTS through its quality-effort graphs, to the extent that it gives a clear indication about which system is the most appropriate for the given scenario;

(b) but, even more important in our vision, it represents the first example of a learning tool for data-transformation systems. In fact, it supports learning in two ways. On the one side, it may help users to approach the adoption of a new system, by presenting them with simple scenarios, asking them to provide some initial specification of the transformation, and dynamically showing how this increases of decreases the quality of the input;

(c) on the other side, it can be used to challenge expert users to find different and more compact ways – i.e., ways that require less effort – to express a transformation without lowering the quality of the efforts.

We believe that this kind of evaluation provides precious insights in the vast and heterogeneous world of transformation systems, and may lead to a better understanding of its different facets. In a more general sense, IQ-METER represents a concrete way to investigate the trade-offs between declarative and procedural approaches, a foundational problem in computer science.

II. OVERVIEW

In this section we introduce the main building blocks of our approach, namely: the quality measure, and the user-effort measure. For further details we refer the reader to [4].

The Quality Measure To achieve a good level of generality, we assume a nested-relational data model for the source and target databases, capable of handling both relational data and XML trees. Given a transformation task, we assume that a "gold standard", has been fixed for each scenario in terms of the output instance expected from the translation. To be able to dynamically update reports during an evaluation session, a key requirement of our approach is the availability of a fast algorithm to measure the similarity of the outputs of a DTS wrt this expected instance. Given the nested relational data model, our instances are trees. Figure 2 shows an example. Indeed, while there are many existing similarity measures for trees, it is important to emphasize that none of these can be used in this framework, for the following reasons:

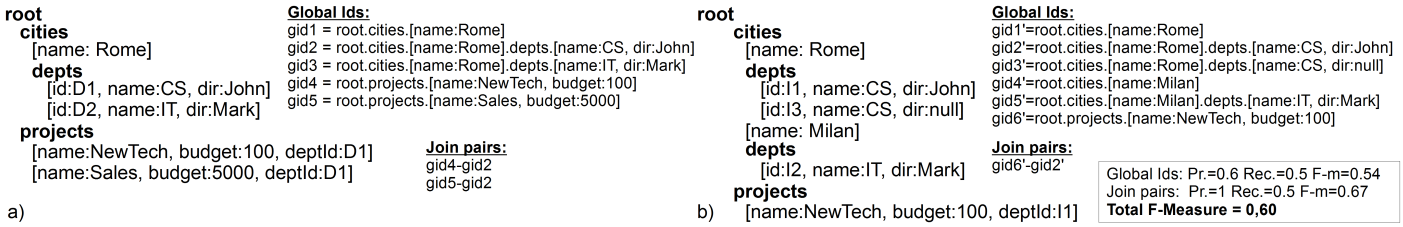

Fig. 2: Comparing Instances

(i) we want to perform frequent and repeated evaluations of each tool, for each selected scenario and mapping specifications of different complexity. Also, we want to be able to work with possibly large instances, to measure how efficient is the transformation generated by a system. As a consequence, we cannot rely on known tree-similarity measures, like, for example, edit distances, that are typically quite expensive;

(ii) the problem above is even more serious, if we think that our instances may be seen as graphs, rather than trees; we need in fact to check key-foreign key references, that can be seen as additional edges, thus making each instance a fully-fledged graph; graph edit distance is knowingly much more complex than tree edit distance;

(iii) even though we were able to circumvent the complexity issues, typical tree and graph-comparison techniques would not work in this setting. To see this, consider that it is typical in mapping applications to generate synthetic values in the output – these values are called *labeled nulls* in data-exchange and *surrogate keys* in ETL. In Figure 2, values $D1, D2, I1, I2, I3$ are of this kind. These values are essentially placeholders used to join tuples, and their actual values do not have any business meaning. We therefore need to check if two instances are identical up to the renaming of their synthetic values.

It can be seen that we face a very challenging task: devising a new similarity measure for trees that is efficient, and at the same time precise enough for the purpose of our evaluation. In order to do this, we introduce the following key-idea: since the instances that we want to compare are not arbitrary trees, but rather the result of a transformation, they are supposed to exhibit a number of regularities; as an example, they are supposedly instances of a fixed nested schema, that we know in advance. This means that we know: (a) how tuples in the instances must be structured; (b) how they should be nested into one another; (c) in which ways they join via key-foreign key relationships.

We design our similarity metric by abstracting these features of the two trees in a set-oriented fashion, and then compare these features using precision, recall and ultimately F-measures to derive the overall similarity. More specifically, for each instance, we compute: (i) first a set of tuple identifiers, also called *local identifiers*, one for each tuple in the instance; (ii) a set of nested tuple identifiers, called *global identifiers*, that capture the nesting relationships among tuples; (iii) a set of pairs of tuple identifiers, called *join pairs*, one for each tuple t_1 that joins a tuple t_2 via a foreign key. Tuple identifiers and join pairs for our example are reported in Figure 2.

It is worth noting that the computation of tuple identifiers requires special care. As it can be seen in the Figure, we keep these values out of our identifiers, in such a way that two instances are considered to be identical provided that they generate the same tuples and the same join pairs, regardless of the actual synthetic values generated by the system.

Based on these ideas, to simplify the treatment we may say that, both for the generated instance and for the expected one, we generate two sets: the set of global identifiers, and the set of join pairs. Then, we compare the respective sets to compute precision and recall, and compute an overall F-Measure that gives us the level of similarity. Figure 2 reports the values of precision and recall and the overall F-measure for our example.

Notice that, in addition to the measure of quality, our algorithm is also capable of providing detailed feedbacks about errors, in terms of missing and extra tuples in the generated output.

Estimating User Efforts To estimate user efforts, we measure the complexity of the mapping specification provided through the DTS GUI. To do this, we model the specification as an *input-graph* with labeled nodes and labeled edges. Experience shows that this model is general enough to cover every data transformation, spanning from schema mapping transformations to ETL ones, and provides more accurate results wrt to previous metrics based on point-and-click counts.

Every element in the source and target schemas is a node in the graph. Arrows among elements in the GUI become edges among nodes in the graph. The tool may provide a library of graphical elements – for example to introduce system functions – that are modeled as additional nodes in the graph. Extra information entered by the user (e.g., manually typed text) is represented as labels over nodes and edges.

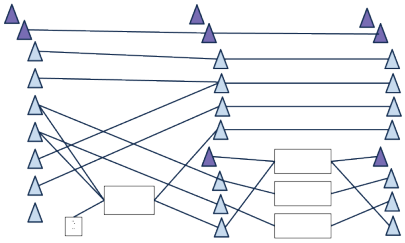

Fig. 3: Input graph for a vertical partition scenario in a commercial mapping system.

We measure the size of such graphs by encoding their elements according to a minimum description length technique [5], and then by measuring the size in bits of such description.

We report in Figure 3 the input-graph for a sample vertical partition scenario expressed using a commercial mapping system. The graph consists of the following elements: (i) 29 nodes for the source and target schemas; notice that the schemas are considered as part of the input and are not counted in the encoding; (ii) there are 5 functions in this scenario that

generate additional nodes; this makes a total of 34 nodes, so that we shall need 6 bits for their encoding; function nodes are labeled by the associated function; based on the size of the library of functions provided by the tool, every function node requires an additional 8 bits for its encoding; (iii) 25 edges (without labels), that we shall encode as pairs of node identifiers (2*6 bits); (iv) finally, to specify the mapping, one of the schema nodes must be labeled by a char, which we need to encode.

The specification complexity is therefore given by the following sum of costs: (5*(6+8)) + (25*(2*6)) + (6+8) = 384 bits.

Designing Scenarios Designing a scenario amounts to choosing a source and target schema, and an input-output function, that is, a set of source instances given as inputs, and a set of target instances considered as expected outputs (the gold standard). It can be seen that the critical point of this process consists in deciding what the expected output should be when some input is fed to the system.

In this respect, our approach is very general, and allows one to select expected outputs in various ways.

(i) it is in principle possible to craft the expected output by hand. However, this can be done only for small scenarios;

(ii) a more typical way – as it was done, for example, in [2] – would be to express the input-output function as a query Q written in a concrete query language, say SQL or XQuery. In this case, for a given input I_S, the expected output would be $I_e = Q(I_S)$;

(iii) as an alternative, one can choose a reference system, design the transformation using the system, compute the result, and take this as the expected output. In many cases can be done, for example, by considering a tool that employs the algorithms in [6], [7], which, given a data-exchange scenario expressed as a set of embedded dependencies, are able to compute the "optimal" solution, i.e., the *core universal solution* [8], in a scalable way.

We want to remark that the system does not impose any of these methods, and all of them are acceptable.

III. Demonstration

In order to get participants as much involved as possible, the demonstration will be centered around a *gamification* of IQ-Meter. In the spirit of evaluating representative systems from different perspectives, we will use the prototype to compare: (i) a schema-mapping research prototypes implementing algorithms of the first-generation [9] and second-generation [10]; (ii) a commercial schema-mapping system; (iii) an open-source ETL tool.

We will select a number of test scenarios of different levels of complexity. Some scenarios will come from previous benchmarks for schema-mappings, like STBenchmark [2]. Other scenarios will come from benchmarks for the ETL world [3]. Finally, we will also consider scenarios from other application domains, like data fusion [11]. To further pursue the investigation of problems that are at the boundaries of different facets of data-transformation domain, we will introduce some new scenarios obtained by enriching existing STBenchmark scenarios with functional dependencies and key constraints [10], in order to make them more challenging and emphasize the relationships between mappings and data-fusion techniques.

In each evaluation session, we will provide a description of the transformation to participants, let them pick-up one or more of the tools under evaluation, and then challenge them to give answers to the following questions:

(a) what is the most effective tool to solve this scenario ?

(b) are you able to correctly express this transformation given a tool you don't know ?

(c) given a correct transformation expressed using tool X, are you able to express the same transformation with the same quality and less effort ?

Finally, users will be asked to design their own scenarios by selecting a source and target schema, input instances and expected output, and run evaluation sessions on them.

During the sessions, users will be presented with instances of various sizes. Thanks to its fast algorithms, IQ-Meter will dynamically compute quality and efforts, and update its report in real time. In addition, it will provide users with detailed explanations for both. As it can be glimpsed in Figure 1, in fact, it provides explanations for the quality measure, in terms of missing and extra tuples in the target, and for the effort measure by showing the computed effort graph. This will show clearly how problems in mapping design can be difficult to debug if proper tools are not adopted.

References

[1] C. Thomsen and T. Bach Pedersen, "ETLDiff: A Semi-automatic Framework for Regression Test of ETL Software," in *DaWaK*, 2006, pp. 1–12.

[2] B. Alexe, W. Tan, and Y. Velegrakis, "STBenchmark: Towards a Benchmark for Mapping Systems," *PVLDB*, vol. 1, no. 1, pp. 230–244, 2008.

[3] A. Simitsis, P. Vassiliadis, U. Dayal, A. Karagiannis, and V. Tziovara, "Benchmarking etl workflows," in *TPCTC*, 2009, pp. 199–220.

[4] G. Mecca, P. Papotti, S. Raunich, and D. Santoro, "What is the IQ of your Data Transformation System?" in *CIKM*, 2012, pp. 872–881.

[5] D. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[6] G. Mecca, P. Papotti, and S. Raunich, "Core Schema Mappings: Scalable Core Computations in Data Exchange," *Information Systems*, vol. 37, no. 7, pp. 677 – 711, 2012.

[7] B. ten Cate, L. Chiticariu, P. Kolaitis, and W. C. Tan, "Laconic Schema Mappings: Computing Core Universal Solutions by Means of SQL Queries," *PVLDB*, vol. 2, no. 1, pp. 1006–1017, 2009.

[8] R. Fagin, P. Kolaitis, and L. Popa, "Data Exchange: Getting to the Core," *ACM TODS*, vol. 30, no. 1, pp. 174–210, 2005.

[9] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin, "Translating Web Data," in *VLDB*, 2002, pp. 598–609.

[10] B. Marnette, G. Mecca, and P. Papotti, "Scalable data exchange with functional dependencies." *PVLDB*, vol. 3, no. 1, pp. 105–116, 2010.

[11] J. Bleiholder and F. Naumann, "Data fusion," *ACM Comp. Surv.*, vol. 41, no. 1, pp. 1–41, 2008.