Schema AND Data: A Holistic Approach to Mapping, Resolution and Fusion in Information Integration

Laura M. Haas[1], Martin Hentschel[2], Donald Kossmann[2], and Renée J. Miller[3]

[1] IBM Almaden Research Center, San Jose, CA 95120, USA
[2] Systems Group, ETH Zurich, Switzerland
[3] Department of Computer Science, University of Toronto, Canada
laura@almaden.ibm.com, martin.hentschel@inf.ethz.ch,
donald.kossmann@inf.ethz.ch, miller@cs.toronto.edu

Abstract. To integrate information, data in different formats, from different, potentially overlapping sources, must be related and transformed to meet the users' needs. Ten years ago, Clio introduced nonprocedural *schema mappings* to describe the relationship between data in heterogeneous schemas. This enabled powerful tools for mapping discovery and integration code generation, greatly simplifying the integration process. However, further progress is needed. We see an opportunity to raise the level of abstraction further, to encompass both data- and schema-centric integration tasks and to isolate applications from the details of how the integration is accomplished. *Holistic information integration* supports iteration across the various integration tasks, leveraging information about both schema and data to improve the integrated result. *Integration independence* allows applications to be independent of how, when, and where information integration takes place, making materialization and the timing of transformations an optimization decision that is transparent to applications. In this paper, we define these two important goals, and propose leveraging *data mappings* to create a framework that supports both data- and schema-level integration tasks.

1 Introduction

Information integration is a challenging task. Many or even most applications today require data from several sources. There are many sources to choose from, each with their own data formats, full of overlapping, incomplete, and often even inconsistent data. To further complicate matters, there are many information integration problems. Some applications require sub-second response to data requests, with perfect accuracy. Others can tolerate some delays, if the data is complete, or may need guaranteed access to data. Depending on the application's needs, different integration methods may be appropriate, but application requirements evolve over time. And to meet the demands of our fast-paced world there is increased desire for rapid, flexible information integration. Many tools

have been created to address particular scenarios, each covering some subset of goals, and some portion of the integration task.

Integration is best thought of not as a single act, but as a process [Haa07]. Since typically the individuals doing the integration are not experts in all of the data, they must first understand what data is available, how good it is, and whether it matches the application needs. Then they must determine how to represent the data in the application, and decide how to standardize data across the data sources. A plan for integrating the data must be prepared, and only then can they move from design to execution, and actually integrate the data. Once the integration takes place, users often discover problems – expected results may be missing, strange results appear – or the needs may change, and they have to crawl through the whole process again to revise it. There are different tools for different (overlapping) parts of the process, as well as for different needs. Figure 1a illustrates the current situation. Information integration is too time-consuming, too brittle, and too complicated. We need to go beyond the status quo, towards a radically simplified process for information integration.

Ten years ago, a new tool for information integration introduced the idea of *schema mappings* [MHH00]. Clio was a major leap forward in three respects. First, it raised the level of abstraction for the person doing the integration, from writing code or queries to creating mappings, from which Clio could generate the code. This higher level of abstraction enabled Clio to support many execution engines from a common user interface [PVM+02]. Second, Clio let users decompose their integration task into smaller pieces, building up complex mappings from simpler ones. Finally, it allowed for iteration through the integration design process, thus supporting an incremental approach to integration. The user could focus first on what they knew, see what mappings were produced, add or adjust, and so on, constantly refining the integration design [FHH+09].

Clio simplified the schema mapping part of the integration process and made it more adaptive. But we need to do more. There is room for improvement in two respects: we need to extend the benefits of a higher level of abstraction to cover both data-centric and schema-centric integration tasks, and we need to make the design phases (and the applications) independent of the actual integration method. We call the first of these *holistic information integration*, and the second *integration independence*.

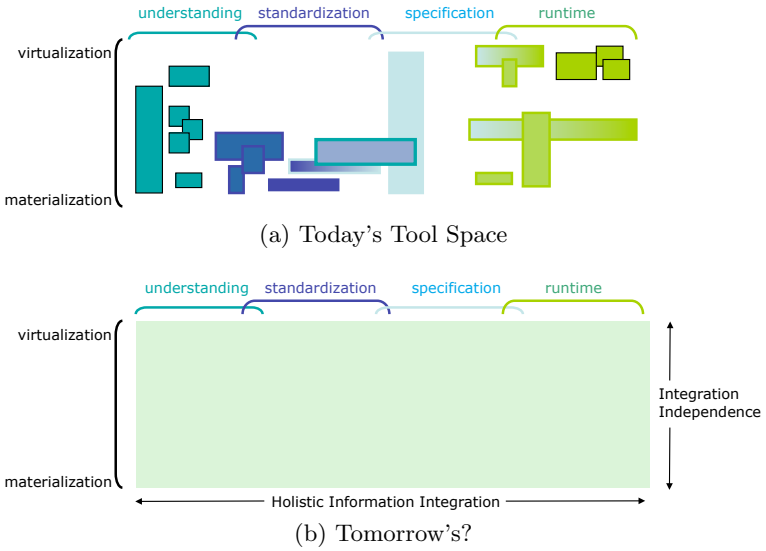
Holistic information integration. Clio only deals with schema-level relationships between a data source and a target (though Clio does data transformation at run-time based on these relationships). Today, other tools are needed to handle data-level integration tasks. Such tasks include entity resolution, which identifies entities in a data source that may represent the same real-world object, and data fusion, which creates a consistent, cleansed view of data from potentially multiple conflicting representations. There is little support for iteration between schema-level and data-level tasks in the integration process. This is unfortunate, because there is no perfect ordering of the tasks. Sometimes, mapping can help with understanding the data and hence with entity resolution and data fusion. But those tasks can also provide valuable information to a mapping

process. By handling both schema and data-level tasks in a common framework, holistically, we hope to enable easier iteration among these phases, and hence, a smoother integration process.

Integration Independence. There are two radically different integration methods: virtualization and materialization. Virtualization (aka, *data integration*) leaves the data where it is, as it is, and dynamically retrieves, merges and transforms it on request. Materialization (*data exchange*) does the integration up front, creating a new data set for requests to run against. Each has its strengths. Virtualization always gets the freshest data, and does no unnecessary work, since the data is integrated only if needed (a lazy form of integration). Materialization often provides better performance, but may process data that will never be requested (an eager approach). Often, the best solution will require a combination of these two approaches. In fact, virtualization cannot solve the whole integration problem today, as we simply do not understand how to do much of integration, including data fusion and entity resolution, virtually. The materialization process handles these data-specific tasks, but it is too heavy duty for some use cases, and a materialization often takes too long to design and build. The decision of which approach to use, and when, must be made early in the integration design process, and, as different integration tools must then be used for the different pieces, is difficult to change. Ideally, applications should be independent of how, when, and where information integration takes place.

Integration independence is analogous to the well-understood concept of data independence. Clio took a large step towards integration independence, by providing a declarative representation of how schemas differ. As a result, applications can be written in a way that is independent of the structural representation of the data. Furthermore, since Clio mappings can be used with either the virtual, data integration, approach or the materialized, data exchange, approach, schema differences may be reconciled either eagerly or lazily. However, current integration engines force the user to choose between the two approaches. For full integration independence, the timing of when structural heterogeneity is reconciled should be an optimization decision that is transparent to applications.

While progress may be made on holistic information integration and integration independence separately, together they hold the potential for truly radical simplification. It would clearly be a leap forward to have a single engine that could move seamlessly between virtualization and materialization, with no changes to the application program [Haa07], and we are currently working towards that goal. However, as long as we continue to need different tools at the design level to handle the schema- and data-specific portions of the integration task, there will always be confusion, overlap, and complexity. If we can, in fact, tackle both schema and data-related integration issues within the same framework, we can use all available information to improve and refine the integration without changing the application. We will be able to move easily among such tasks as understanding, mapping, fusion, and entity resolution, and even to execution and back. It will enable us to handle the ever-changing dynamics of application needs for performance, completeness, and accuracy, and to react



(a) Today's Tool Space

(b) Tomorrow's?

Fig. 1: Effect of Holistic Information Integration and Integration Independence

quickly to data and schema evolution. Rapid prototyping and what-if scenarios will be more effectively supported. We expect that a unified framework will also reduce the knowledge needed by the integrator – of different tools, schemas and the data itself. Holistic information integration and integration independence together can lead to the simplicity of Figure 1b.

This paper is organized as follows. In the next section we describe some foundational work. Section 3 proposes leveraging data mappings to extend the benefits of nonprocedural mappings to the data level. We illustrate the benefits and the challenges through a detailed example. Finally, we conclude with some thoughts on next steps and our current work in Section 4.

2 Foundations: Schema and Data Mapping

Up until ten years ago, most metadata management research focused on the schema matching problem, where the goal was to discover the existence of possible relationships between schema elements. The output of matching was typically modeled as a relation over the set of elements in two schemas (most often as a set of attribute pairs) [RB01]. Often such work was agnostic as to the semantics of the discovered relationships. At best, a matching had very limited transformational power (for example, a match might only allow copying of data, but no joins or complex queries). Indeed this feature was viewed as a virtue as it enabled the development of generic matchers that were independent of a specific data model.

However, the last decade has shown how important the semantics of these relationships are. During this period, we have made remarkable progress, due

to the development and wide-spread adoption of a powerful declarative schema mapping formalism with a precise semantics. Clio [HMH01] led the way in both developing this formalism and in providing solutions for (semi)-automatically discovering, using and managing mappings. The benefits of considering semantics are clear. First, having a common agreement on a robust and powerful transformation semantics enables the exploitation of schema mappings for both virtual and materialized integration. Second, schema mapping understanding and debugging tools rely on this semantics to help elicit nuanced details in mappings for applications requiring precise notions of data correctness. Third, having a widely adopted semantics has enabled a large and growing body of research on how to manage schema mappings, including how to compose, invert, evolve, and maintain mappings. Indeed, schema mappings have caused a fundamental change in the research landscape, and in the available tools.

2.1 Schema Mappings

Informally, schema mappings are a relationship between a query over one schema and a query over another. A query can be as simple as an expression defining a single concept (for example, the set of all clients) and the relationship may be an *is-a* or containment relationship stating that each member of one concept *is-a* member of another. We will use the arrow $\rightarrow$ to denote an *is-a* relationship, e.g., `Client -> Guest`. Since queries can express powerful data transformations, complex queries can be used to relate two concepts that may be represented completely differently in different data sources.

To precisely define the semantics of a schema mapping, Clio adapted the notion of *tuple-generating dependencies* or referential constraints from relational database theory [BV84]. A schema mapping is then a *source-to-target tuple-generating dependency* from one schema to another (or in the case of schemas containing nesting, a nested referential constraint) [PVM$^+$02]. Such constraints (which express an *is-a* or containment relationship) were shown to have rich enough transformational power to map data between complex independently-created schemas. Furthermore, this semantics was useful in not only (virtual) data integration [YP04], but it also fueled the development of a new theory of data exchange [FKMP05]. This theory provides a foundation for materialized information integration and is today one of the fastest growing areas in integration research.

Because Clio mappings have the form $Q(S) \rightarrow Q(T)$, they are declarative and independent of a specific execution environment. Early in its development, Clio provided algorithms for transforming mappings into executable data exchange programs for multiple back-end integration engines [PVM$^+$02]. Specifically, Clio mappings can be transformed into executable queries (in SQL or Xquery), XSLT scripts, ETL scripts, etc. This is one of the key aspects to Clio's success as it freed application writers from having to write special-purpose code for navigating and transforming their information for different execution environments.

In addition, this clean semantics forms the foundation for a new generation of user front-ends that support users developing applications for which the

correctness of the data (and hence, of the integration) is critical. Tools such as data-driven mapping GUIs [YMHF01,ACMT08] help users understand, and possibly modify, what a mapping will do by showing carefully chosen examples from the data. Likewise, tools for debugging mappings [CT06,BMP+08] help a user discover how mappings have created a particular (presumably incorrect) dataset. Visual interfaces like Clip [RBC+08] permit users to develop mappings using a visual language. There has also been a proliferation of industry mapping tools from companies including Altova, IBM, Microsoft and BEA. The existence of a common mapping semantics has enabled the development of the first mapping benchmark, STBenchmark [ATV08], which compares the usability and expressibility of such systems.

2.2 Data Mappings

Schema mappings permit data under one schema to be transformed into the form of another. However, it may be the case that two schemas store some of the same information. Consider a simple schema mapping that might connect two hotel schemas:

```
M:  Client -> Guest
```

Given a Client tuple c, this mapping states that c is also a Guest tuple. However, we may want to assert something stronger. We may know that c actually represents the same real world person as the Guest tuple g. (For example, entity resolution techniques can be used to discover this type of relationship.) Ideally, we'd like to be able to make the assertion: c `same-as` g, as an ontology language such as OWL would permit.

This is a common problem, so much so that it has been studied not only in ontologies, but also in relational systems where the data model does not provide primitives for making `same-as` assertions and where there is a value-based notion of identity. Kementsietsidis et al. [KAM03,KA04] explored in depth the semantics of data mappings such as this. They use the notion of *mapping tables* to store and reason about sets of data mappings. Mapping tables permit the specification of two kinds of data mappings, `same-as` and `is-a`. If c `same-as` g, then any query requesting information about client c will get back data for guest g as well, and vice versa. However, for the latter, if c `is-a` g, then for queries requesting information about g the system will return c's data as well, but queries requesting c will not return values from g. A given mapping table can be declared to have a closed-world semantics meaning that *only* the mappings specified in the table are permitted. This is a limited form of negation which we will discuss further in the next section.

2.3 Mapping Discovery

Clio pioneered a new paradigm in which schema mapping creation is viewed as a process of query discovery [MHH00]. Given a matching (a set of correspondences) between attributes in two schemas, Clio exploits the schemas and their

constraints to generate a set of alternative mappings. Detailed examples are given in Fagin et al. [FHH+09] . In brief, Clio uses logical inference over schemas and their constraints to generate all possible associations between source elements (and all possible associations between target elements) [PVM+02]. Intuitively, Clio is leveraging the semantics that is embedded in the schemas and their constraints to determine a set of mappings that are consistent with this semantics.

Since Clio laid the foundation for mapping discovery, there have been several important advances. First, An et al. [ABMM07] showed how to exploit a conceptual schema or ontology to improve mapping discovery. Their approach requires that the relationship of the conceptual schema to the schemas being mapped is known. They show how the conceptual schema can then be used to make better mapping decisions.

An interesting new idea is to use data mappings (specifically *same-as* relationships) to help in the discovery of schema mappings. Suppose we apply an entity-resolution procedure to tuples (entities) stored under two schemas to be mapped. We then also apply a schema mapping algorithm that postulates a set of possible mappings. For a given schema mapping $m : A \rightarrow B$, suppose further that mapping m implies that two entities (say e_1 from A and e_2 from B) must be the same entity (this may happen if e_1 and e_2 share a key value). If the similarity of e_1 and e_2 is high, then the entity-resolution procedure will likely come to the same conclusion, agreeing with the schema mapping algorithm. This should increase the confidence that mapping m is correct. If however, e_1 and e_2 are dissimilar, then this should decrease confidence in the mapping m. This is the basic idea behind Iliads [UGM07]. Evidence produced by entity-resolution is combined with evidence produced by schema mapping using a concept called *inference similarity*. This work showed that combining the statistical learning that underlies entity-resolution algorithms with the logical inference underlying schema mapping discovery can improve the quality of mapping discovery.

Iliads is a step towards our vision for holistic information integration. As we explore in the next section, there is much more that can be done.

3 A Holistic Approach to Information Integration

We would like to bring to the overall information integration process the benefits of a higher level of abstraction and a unified framework. We envision a holistic approach, in which all integration tasks can be completed within a single environment, moving seamlessly back and forth between them as we refine the integration. A key element in achieving this vision will be *data mappings*. In this section, we define this concept, and illustrate via an example how data mappings enable holistic information integration.

3.1 Our Building Blocks

By analogy to schema mappings, a data mapping defines a relationship between two data elements. It takes the form of a rule, but rather than identifying the

(ID)	Name	Home	Income	TotalSpent	Comps
@GuestRM	Renée Miller	Toronto	1.3M	250K	Champagne
@GuestLA	Laurence Amien	Toulouse	350K	75K	None
@GuestDK	Donald Kossmann	Munich	575K	183K	Truffles
@GuestLH	Laura Haas	San Jose	402K	72K	None

Table 1: Las Vegas schema for Guest and sample data

(ID)	Prénom	Nom	Ville	Logements	Casino	RV	Cadeau
@ClientRM	René	Miller	Toronto	300	10K	100K	rien
@ClientLA	Laurence	Amiens	Toulouse	5K	250K	350K	chocolate
@ClientDK	Donald	Kossmann	Munich	15K	223K	575K	truffles
@ClientMH	Martin	Hentschel	Zurich	10K	95K	250K	bicycle
@ClientLH	Laura	Haas	San Jose	1K	50K	402K	rien

Table 2: French schema for Client and sample data

data it refers to by that data's logical properties (as would a schema mapping), it uses object identifiers to refer directly to the data objects being discussed. A data mapping, therefore, relates together two objects. The simplest relationship we can imagine might be *same-as*, e.g., *Object34 same-as ObjectZ18* (where Object34 and ObjectZ18 are object identifiers in some universe). Data mappings could be used for specifying the results of entity resolution, or as part of data fusion.

It is not enough to add such rules; we also need an integration engine that can work with both data mappings and schema mappings, and allow us to move seamlessly from integration design to integration execution and back again. We are currently building such an engine, exploiting a new technique that interprets schema mappings at integration runtime [HKF+09]. Conceptually, as the engine sees data objects in the course of a query, it applies any relevant rules (schema or data mappings) to determine whether the objects should be returned as part of the data result. Enhancements to improve performance via caching, indexing, pre-compiling, etc., can be made, so that the engine provides integration independence as well. This in turn enables a single design environment. In this paper, we assume the existence of such an engine, without further elaboration.

3.2 Holistic Information Integration: An Example

Suppose a casino in Las Vegas has just acquired a small casino in France. The management in Las Vegas would like to send a letter to all the "high rollers" (players who spend large amounts of money) of both casinos, telling them the news, and inviting them to visit. They do not want to wait a year while the two customer records management systems are integrated. Fortunately, they have available our new integration engine. Jean is charged with doing the integration.

Table 1 and Table 2 show the existing (highly simplified) schemas, and a subset of data, for the Las Vegas and French customer management systems,

respectively. Jean's first step is to define "high roller". To this end, she creates the following rules:

```
Client [Logements+Casino > 100K] -> HighRoller
Guest [TotalSpent > 100K] -> HighRoller
```

The above syntax is used for illustration only. The first rule says that when we see a Client object, where the lodging plus the casino fields total more than 100K, then that Client is a high roller – it should be returned whenever HighRoller's are requested. Likewise, the second says that Guests whose TotalSpent is over 100K are also HighRollers. Such rules can be easily expressed in most schema mapping rule languages. With these two rules, it is possible to enter a query such as "Find HighRollers" (this might be spelled `//HighRoller` in XQuery, for example), with the following results:

Guest: [Renée Miller, Toronto, 1.3M, 250K, Champagne]
Guest: [Donald Kossmann, Munich, 575K, 183K, Truffles]
Client: [Laurence, Amiens, Toulouse, 5K, 250K, 350K, chocolats]
Client: [Donald, Kossmann, Munich, 15K, 223K, 575K, truffles]
Client: [Martin, Hentschel, Zurich, 10K, 95K, 250K, bicycle]

Note that a mixture of Guests and Clients are returned, since there has been no specification of an output format. We believe that this type of tolerance of heterogeneity is important for a holistic integration system, as it preserves information and allows for later refinement of schema and data mappings.

Jean notices that there are two entries for Donald Kossmann, one a "Guest", from the Las Vegas database, and the other a "Client" from the French one. She decides they are the same (they come from the same town, receive the same gift, etc). She only wants to send Donald one letter, so she'd like to ensure that only one entry comes back for him. Ideally, she would just specify a rule saying that the guest and client Donald Kossmann are the same.

We enable Jean to do this by the following rule (again, syntax is for illustration only):

```
@GuestDK <- @ClientDK
```

where the two sides represent the "addresses" or unique ids of the two objects she wants to equate. This rule says that the guest Donald Kossmann is really the same as the client, and that the merge of the two nodes should be returned, with the Client fields being added to the Guest. In other words, the Client object is merged into the Guest object, creating an asymmetric *merge into* semantics. Other semantics are definitely possible. For example, the objects could be merged into a new object (symmetric *merge* semantics), or not merged at all, but treated as one during query processing (*equivalence* semantics) so that only one is returned. From an implementation perspective, *merge into* is simpler to model, and seems to offer sufficient power for the scenarios we have worked with so far, but more investigation is clearly needed. With this new rule, Jean gets the following query result:

Guest: [Renée Miller, Toronto, 1.3M, 250K, Champagne]
Guest: [Donald Kossmann, Munich, 575K, 183K, Truffles, Donald, Kossmann, Munich, 15K, 223K, 575K, truffles]
Client: [Laurence, Amiens, Toulouse, 5K, 250K, 350K, chocolats]
Client: [Martin, Hentschel, Zurich, 10K, 95K, 250K, bicycle]

Note that Donald is only returned once, as a `Guest`, but with all the fields of both guests and clients, preserving all the information associated with the object.

With a simple query and just a few schema and data mapping rules, Jean has found the high rollers and all the available information about each. Or has she? Seeing Donald in both lists reminds her that there could be customers who have visited both casinos, not spending enough in either to qualify as high rollers, but in total spending across the two casinos clearly qualifying. She would like to add these folks, if any, to the list. This requires entity resolution to detect when two entities (here casino visitors) are the same. Many algorithms for entity resolution exist; most do some form of clustering of records, often with user input on which fields are important, or how to measure similarity. Jean runs such an algorithm, and accepts the results when they are shown to her, creating the following additional rules.

```
@GuestLA <- @ClientLA
@GuestLH <- @ClientLH
@GuestRM <- @ClientRM
```

She then can add her new schema mapping rule, to wit:

```
Guest [TotalSpent+Logements+Casino > 100K] -> HighRoller
```

`Find HighRollers` now returns:

Guest: [Renée Miller, Toronto, 1.3M, 250K, Champagne, René, Miller, Toronto, 300, 10K, 100K, rien]
Guest: [Donald Kossmann, Munich, 575K, 183K, Truffles, Donald, Kossmann, Munich, 15K, 223K, 575K, truffles]
Guest: [Laura Haas, SJ, 402K, 72K, None, Laura, Haas, SJ, 1K, 50K, 402K, rien]
Guest: [Laurence Amien, Toulouse, 350K, 75K, None, Laurence, Amiens, Toulouse, 5K, 250K, 350K, chocolats]
Client: [Martin, Hentschel, Zurich, 10K, 95K, 250K, bicycle]

Jean is happy with this result; now she wants to transform it into a simple form for the two casinos to use. At this point, she is more familiar with the data, so she can create an output schema and map the Guest and Client schemas to that. She does this by replacing our earlier, simple mapping rules by the refined versions in Figure 2. This pair of rules not only specify that Clients and Guests that spend a certain amount are HighRollers, but tells how to construct a High Roller instance from a Client or Guest instance. Note that the Guest rule, in concert with the earlier data mappings, completes data fusion, by telling how the various fields from the merged objects should be reconciled. For example,

```
Client [Logements+Casino > 100K] as $c -> <HighRoller>
                    <FullName>$c.Prenom ||$c.Nom </FullName>
                    <City>$c.Ville </City>
                    <Spent>$c.Logements + $c.Casino </Spent>
                    <Gift>$c.Cadeau </Gift>
                </HighRoller>

Guest [TotalSpent+Logements+Casino > 100K] as $g ->   <HighRoller>
                    <FullName>$g.Name </FullName>
                    <City>$g.Home </City>
                    <Spent>$g.TotalSpent + $g.Logements + $g.Casino </Spent>
                    <Gift>$g.Comps || $g.Cadeau</Gift>
                </HighRoller>
```

Fig. 2: Mapping Rules

the `Spent` field of HighRoller is defined to be the sum of all the fields that have anything to do with spending in Guest (+ the merged Client) objects. The `Gift` field is defined as the concatenation of the `Comps` and `Cadeau` fields for simplicity; Jean could, of course, have used a fancier rule to resolve the Gift values, for example, preferring a value other than "Rien" or "None", or choosing one gift based on its monetary value.

Now if Jean runs the query again, with these new rules, her result would be:

HighRoller: [Renée Miller, Toronto, 260.3K, Champagne rien]
HighRoller: [Donald Kossmann, Munich, 421K, Truffles truffles]
HighRoller: [Laurence Amien, Toulouse, 330K, None chocolats]
HighRoller: [Laura Haas, SJ, 123K, None rien]
HighRoller: [Martin Hentschel, Zurich, 105K, bicycle]

The integration is now ready to use. These results could be saved in a warehouse for reference, or the query could be given to the two casinos to run as needed, getting the latest, greatest information. This in itself is a major advance over the state of the art, where totally different design tools and runtime engines would be used depending on whether the goal was to materialize or federate (provide access to the virtual integration). Further, Jean was able to do this with minimal knowledge of the French schema, leveraging the mapping rules, the data, and the flexibility to iterate. The two types of rules work well together. Schema mapping rules gather the data; they can be used to transform it when ready. Data mapping rules record decisions on which entities are the same, and ensure that the query results contain all available information about each entity.

Another benefit of this holistic integration approach is that data-level and schema-level operations can be interwoven. In our example, defining some simple schema-level mappings between Guest and Client (e.g., `Client/(Prénom || Nom) -> Guest/Name` might make it easier to do comparisons for entity resolu-

tion. However, if we've done entity resolution and can observe that for each pair that we've found, the Client RV field is the same as the Guest Income field, we may be able to guess that RV (for *revenu*) should be mapped to Income if we wanted that value.

Of course, life is not this simple, and we need to explore what cases our holistic framework should handle. Continuing our example, let's suppose that René Miller visits the French casino again, and an alert clerk notes that René is a guy, while Renée is a woman's name. Not wishing to waste champagne on the wrong person, he investigates, and discovers that this is, indeed, a different person, although both are from Toronto. Thus the rule

```
@GuestRM <- @ClientRM
```

is wrong, and must be removed. However, without changes to the entity resolution logic, it is quite possible that such a rule would be re-produced sometime in the future, and no one would notice. In addition to the champagne issue, it could be dangerous financially to extend to Mr. Miller the type of credit that Ms. Miller legitimately enjoys. Hence, it would be useful to be able to have negative data mapping rules, i.e.,

```
@GuestRM !<-! @ClientRM
```

Where `!<-!` means "under no circumstances merge these entities", here the client entity René Miller with the guest entity Renée Miller. Such rules seem quite useful, but adding negation into rule languages has typically proven to add complexity to query processing. We need to understand whether this very specific form of negation causes similar problems.

3.3 Further Opportunities

While the above example shows the immediate value that could be provided by data mappings, we believe that the concept will enable new tools that can provide further value. An obvious place to start is with discovering various types of data mappings. Entity resolution essentially discovers *same-as* relationships today, and data mappings allow us to harness that power and include it within our holistic framework. But other types of relationships between entities are possible, and can be useful for the integration process. For example, understanding *part-of* relationships can help with schema mapping. The linked open data community is providing typed links between objects, where the types may come from a data model or an ontology, specifying any type of relationship. Specialized discovery tools for certain domains and relationships could be valuable, as semantics of those constructs could be leveraged [HXK+09]. For example, the *same-as* relationship between genes is quite different than *same-as* between people.

Along similar lines, we may consider generalizing the notion of schema mappings, which today focus on *contained-in* relationships. There may be other types of schema-level relationships we may be able to discover that could aid the integration process. Meanwhile, the principles of open linked data include not only

using URIs, but also providing useful information when someone looks up a URI or dereferences an HTTP URI. Clearly mappings, data and schema, can be a key to providing semantically relevant information.

4 Conclusions

In this paper, we have argued that *holistic information integration* and *integration independence* are important, inter-related goals for research in information integration. Ten years ago we took a big step towards integration independence by enriching our modeling capabilities with schema-level mappings. That gave us a nonprocedural expression of the differences between schemas, allowing us to produce code to reconcile those differences automatically, for different integration engines, whether a data integration engine using virtualization, or an engine for data exchange that uses materialization. However, the engines remained distinct, with differing capabilities. Further, the schema and data worlds have for the most part been considered independently, forcing separate tools to be developed for each, and fragmenting the integration design process. Hence, applications have continued to be impacted by the choice of integration methods, and users have been baffled by the variety of tools.

This paper proposed a step towards holistic information integration. By adding *data mappings*, we enable both schema and data issues to be addressed within a single integration framework, opening the door to new tools, and a more iterative approach to integration. Still, much work remains to be done. It is not trivial to build an integration engine that can move easily between virtualization and materialization of integrated data, especially one that can also deal with the implications of data mappings. Algorithms to handle typical data-level tasks such as data fusion and entity resolution must be made efficient and effective during data integration, when the end result will not be materialized. Research is also needed on the semantics, limits and types of data mappings, and on tools that leverage these mappings to make the integration task easier. These are doubtless just a few of the challenges ahead, on our path to integration independence and holistic information integration.

References

[ABMM07] Y. An, A. Borgida, R. J. Miller, and J. Mylopoulos. A Semantic Approach to Discovering Schema Mapping Expressions. In *IEEE ICDE Conf.*, pages 206–215, 2007.

[ACMT08] B. Alexe, L. Chiticariu, R. J. Miller, and W.-C. Tan. Muse: Mapping Understanding and deSign by Example. In *IEEE ICDE Conf.*, pages 10–19, 2008.

[ATV08] B. Alexe, W.-C. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *Proceedings of the VLDB Endowment*, 1(1):230–244, 2008.

[BMP+08] A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa. Schema Mapping Verification: The Spicy Way. In *EDBT Conf.*, pages 85–96, 2008.

[BV84] C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *Journal of the ACM*, 31(4):718–741, 1984.

[CT06] L. Chiticariu and W.-C. Tan. Debugging Schema Mappings with Routes. In *VLDB Conf.*, pages 79–90, 2006.

[FHH+09] R. Fagin, L. M. Haas, M. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema Mapping Creation and Data Exchange. In A. T. Borgida, V. K. Chaudhri, P. Giorgini, and E. S. Yu, editors, *Conceptual Modeling: Foundations and Applications, Essays in Honor of John Mylopoulos*, volume 5600. Springer, 2009.

[FKMP05] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, 336(1):89–124, May 2005. Extended version of ICDT 2003.

[Haa07] L. M. Haas. Beauty and the Beast: The Theory and Practice of Information Integration. *Lecture Notes in Computer Science*, 4353, 2007. ICDT Conf.

[HKF+09] M. Hentschel, D. Kossmann, D. Florescu, L. Haas, T. Kraska, and R. J. Miller. Scalable Data Integration by Mapping Data to Queries. Technical Report 633, ETH Zurich, Systems Group, Dept. of Computer Science, 2009.

[HMH01] M. A. Hernández, R. J. Miller, and L. M. Haas. Clio: A Semi-Automatic Tool For Schema Mapping. In *ACM SIGMOD Conf.*, page 607, 2001. System Demonstration.

[HXK+09] O. Hassanzadeh, R. Xin, A. Kementsietsidis, L. Lim, R. J. Miller, and Min Wang. Linkage Query Writer. In *VLDB Conf.*, 2009. System Demonstration.

[KA04] A. Kementsietsidis and M. Arenas. Data Sharing Through Query Translation in Autonomous Sources. In *VLDB Conf.*, pages 468–479, 2004.

[KAM03] A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. *ACM SIGMOD Conf.*, 32(2):325–336, 2003.

[MHH00] R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *VLDB Conf.*, pages 77–88, 2000.

[PVM+02] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating Web Data. In *VLDB Conf.*, pages 598–609, 2002.

[RB01] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10(4):334–350, 2001.

[RBC+08] A. Raffio, D. Braga, S. Ceri, P. Papotti, and M. A. Hernández. Clip: a Visual Language for Explicit Schema Mappings. In *IEEE ICDE Conf.*, pages 30–39, 2008.

[UGM07] O. Udrea, L. Getoor, and R. J. Miller. Leveraging Data and Structure in Ontology Integration. In *ACM SIGMOD Conf.*, pages 449–460, 2007.

[YMHF01] L.-L. Yan, R. J. Miller, L. Haas, and R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. *ACM SIGMOD Conf.*, 30(2):485–496, 2001.

[YP04] C. Yu and L. Popa. Constraint-Based XML Query Rewriting For Data Integration. *ACM SIGMOD Conf.*, 33(2):371–382, 2004.