Why Your Data Won't Mix: Semantic Heterogeneity

Alon Y. Halevy
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
alon@cs.washington.edu

1. INTRODUCTION

When database schemas for the same domain are developed by independent parties, they will almost always be quite different from each other. These differences are referred to as semantic heterogeneity. Semantic heterogeneity also appears in the presence of multiple XML documents, web services and ontologies – or more broadly, whenever there is more than one way to structure a body of data. The presence of semi-structured data exacerbates semantic heterogeneity, because semi-structured schemas are much more flexible to start with. In order for multiple data systems to cooperate with each other, they must understand each other's schema. Without such understanding, the multitude of data sources amounts to a digital version of the Tower of Babel.

This article begins by reviewing several common scenarios in which resolving semantic heterogeneity is crucial for building data sharing applications. We then explain why resolving semantic heterogeneity is difficult, and review some recent research and commercial progress in addressing the problem. Finally, we point out the key open problems and opportunities in this area.

2. SCENARIOS OF SEMANTIC HETEROGENEITY

Enterprise Information Integration (EII): Enterprises today are increasingly facing data management challenges that involve accessing and analyzing data residing in multiple sources, such as database systems, legacy systems, ERP systems and XML files and feeds. For example, in order for an enterprise to obtain a "single view of customer", they must tap into multiple databases. Similarly, to present a unified external view of their data, either to cooperate with a third party or to create an external facing web site, they must access multiple sources. As the electronic marketplace becomes more prevalent, these challenges are becoming bottlenecks in many organizations.

There are many reasons for which data in enterprises resides in multiple sources in what appears to be a haphazard fashion. First, many data systems were developed independently for targeted business needs, but when the business needs changed, data needs to be shared between different parts of the organization. Second, enterprises acquire many data sources as a result of mergers and acquisitions.

Over the years, there have been multiple approaches to addressing enterprise information integration challenges. Until the late 90's, the two leading approaches were data warehousing and building custom solutions. Data warehousing solutions had the disadvantage of accessing stale data in many cases and not being able to work across enterprise boundaries. Custom code solutions are expensive, hard to maintain, and typically not extensible.

In the late 90's, several companies offered solutions that queried multiple data sources in real-time. In fact, the term EII is typically used to refer to these solutions. While the users of these systems still see a single schema (whether relational or XML), queries are translated on the fly to appropriate queries over the individual data sources, and results are combined appropriately from partial results obtained from the sources. Consequently, answers returned to the user are always based on fresh data. Interestingly, several of these companies built their products on XML platforms, because the flexibility of XML (and more generally of semi-structured data) deemed it more appropriate for data integration applications. A recent article [8] surveys some of the challenges faced by this industry. Finally, more recent research has proposed peer-to-peer architectures for sharing data with rich structure and semantics [1].

In any of these data sharing architectures, reconciling semantic heterogeneity is key. No matter whether the query is issued on the fly, or data is loaded into a warehouse, or whether data is shared through web services or in a peer-to-peer fashion, the semantic differences between data sources need to be reconciled. Typically, these differences are reconciled by *semantic mappings*. These are expressions that specify how to translate data from one data source into another in a way that preserves the semantics of the data, or alternatively, reformulate a query posed on one source into a query on another source. Semantic mappings can be specified in a variety of mechanisms, including SQL queries, XQuery expressions, XSLT scripts, or even Java code.

In practice, the key issue is the amount of effort it takes to specify a semantic mapping. In a typical data integration scenario, over half of the effort (and sometimes up to 80%) is spent on creating the mappings, and the process is labor intensive and error prone. Today, most EII products come with some tool for specifying these mappings, but the tools are completely manual – an expert needs to specify the exact

mapping between the two schemas.

Querying and Indexing the Deep Web: The *deep web* refers to web content that resides in databases and is accessible behind forms. Deep web content is typically not indexed by search engines because the crawlers that these engines employ cannot go past the forms. In a sense, the form can be seen as a (typically small) schema, and unless the crawler can understand the meaning of the fields in the form, it gets stuck there.

The amount and value of content on the deep web are spectacular. By some estimates, there are 1-2 orders of magnitude more content on the deep web than the surface web. Examples of such content range from classified ads in thousands of newspapers across the world, to data in government databases, product databases, university repositories and more.

Here too, the challenge stems from the fact that there is a very wide variety in the way web-site designers model aspects of a given domain. Therefore, it is impossible for designers of web crawlers to assume certain standard form-field names and structures as they crawl. Even in a simple domain such as searching for used cars, the heterogeneity in forms is amazing. Of course, the main challenge comes from the scale of the problem. For example, the web site at www.everyclassified.com, the first site to aggregate content from thousands of form-based sources, includes over 5000 semantic mappings of web forms in the common categories of classified ads. Later in the article, I will describe the ideas which made this web site possible.

It is important to emphasize that accessing the deep web is even more of a challenge for the content providers than it is for the search engines. The content providers thrive on getting users' attention. In the early days of the WWW, any good database would be immediately known (e.g., IMDB for movies). However, the number of such databases today is vast (estimated in the hundreds of thousands), and people do not know about them. Instead, people's searches start from the search box of their favorite engine, and these engines do a very poor job of indexing deep web content. Hence, if I create an excellent database of middle-eastern recipes and put it on the web behind a form, it may remain invisible. Ironically, I'm better off creating a set of web pages with my recipe contents than creating an easily searchable database. Finally, it should be noted that enterprise search faces a somewhat similar problem: many of the interesting data sources within an enterprise are in databases, and even providing simple keyword search over this content is quite challenging.

Merchant Catalog Mapping: A frequent example of semantic heterogeneity occurs in aggregating product catalogs. Consider an online retailer such as Amazon.com. Such a retailer accepts feeds of products from thousands of merchants, each trying to sell their goods online. To aggregate the vast number of feeds, online retailers prescribe a schema: a hierarchy of products and their associated properties. To sell their products online, the merchants need to send a feed that adheres to the prescribed schema. However, on the back-end, the data at the merchant is stored in their local schema, which is likely quite different from the one prescribed by the retailer (and typically covers a small fragment of that schema). Hence, the problem we face here is creating mappings between thousands of merchants and a growing number of recognized online retailers (roughly 10 of them in the USA at this time). An interesting point to note about this scenario is that there is not necessarily a *single* correct semantic mapping from the merchant's schema to that of the retailer. Instead, because there are subtle differences between product categories, and products can often be mapped to several categories, there are multiple mappings that may make sense, and the "best" one is the one that ultimately sells more products!

Schema versus Data heterogeneity: Heterogeneity occurs not only in the schema, but also in the actual data values themselves. For example, there may be multiple ways of referring to the same product. Hence, even though you are told that a particular field in a merchant's data maps to ProductName, that may not be enough to resolve multiple references to a single product. As other common examples, there are often multiple ways of referring to companies (e.g., IBM vs. International Business Machines), people names (that are often incomplete), and addresses. To fully integrate data from multiple sources one needs to handle both the semantic-level heterogeneity and the data-level heterogeneity. Typically, different products have addressed these two parts of the problem in isolation. As one example, several of the products for 'global spend analysis' have focused on data-level heterogeneity. This article focuses mostly on schema heterogeneity.

Schema heterogeneity and semi-structured data: I argue that the problem of semantic heterogeneity is exacerbated when we deal with semi-structured data, for several reasons. First, the applications involving semi-structured data are typically ones that involve sharing data among multiple parties, and hence semantic heterogeneity is part of the problem from the start. Second, schemas for semi-structured data are much more flexible, so we are more likely to see variations to the schema. Finally, the main advantage of semi-structured is that attributes can be added to the data at will (or even simply derived from inspecting the data itself), and once that flexibility is in place, the number of additional attributes we see is significant, and understanding their exact meaning becomes crucial. On the flip side, in many applications involving semi-structured data it is enough to reconcile only a specific set of attributes, while we can still manipulate and display any other attribute. Specifically, we only need to reconcile those attributes that are going to be used for equating data across multiple sources.

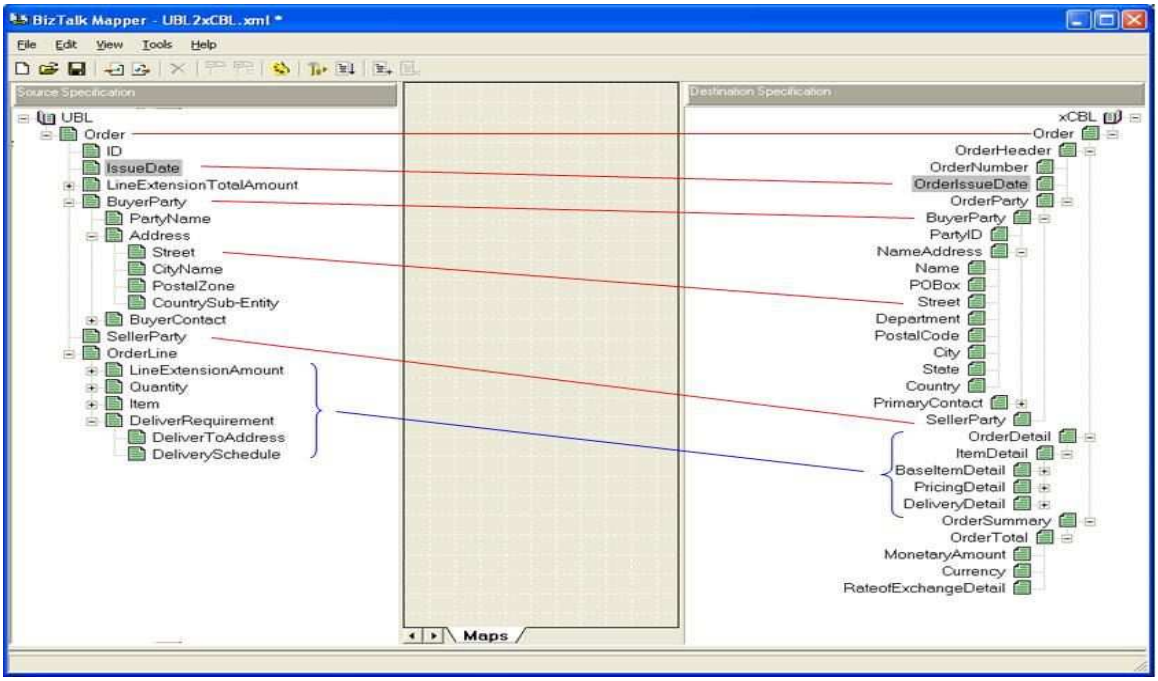

Figure 1: Reconciling two disparate schemas.

3. WHY IS IT SO HARD?

The problem of reconciling schema heterogeneity has been a subject of research for decades, but solutions are few. The fundamental reason that makes semantic heterogeneity so hard is that the data sets were developed independently, and therefore varying structures were used to represent the same or overlapping concepts. In many cases, we are trying to integrate data systems that were developed for slightly (or vastly) different business needs. Hence, even if they model overlapping domains, they will model them in different ways. Differing structures are a byproduct of human nature – people think differently from one another even when faced with the same modeling goal. As a simple illustration, one of the assignments I give in my senior-level database course is to design an inventory schema based on a one-page English description of what it should cover. Invariably, the schemas I get from my students are vastly different [6].

From a practical perspective, one of the reasons that schema heterogeneity is difficult and time consuming is that it requires both domain and technical expertise: you need a person that understands the business meaning of each of the schemas being reconciled and people skilled in writing transformations (e.g., SQL or XQuery experts).

While schema heterogeneity is challenging for humans, it is drastically more challenging for programs. A program is only given the two schemas to reconcile – but those schemas are merely symbols. They do not capture the entire meaning or intent of the schemas – those are only in the minds of the designers.

Figure 1 illustrates some of the challenges in resolving semantic heterogeneity. The figure shows a typical manual schema matching tool in which the designer needs to draw lines between the matching attributes of the two schemas. As can be seen in the example, there are several kinds of semantic discrepancies between schemas: (1) the same schema element in the two schemas are given different names (e.g., IssueDate and OrderIssueDate), (2) attributes in the schema are grouped into table structures (or XML nesting) in different ways (e.g., consider the subtrees of the BuyerParty element in the two schemas), and (3) one schema may cover aspects of the domain that are not covered by the other (e.g., the left schema does not have anything like OrderSummary in the right schema).

When reconciling heterogeneity from thousands of web forms, there are additional sources of heterogeneity. Some forms are already specialized for a particular domain (used cars, jobs), while in others the user needs to select a category before entering additional attributes. In some cases, the location is already implicit in the form (e.g., using some hidden fields), while in others the user needs to select a city and state or zip.

It is often argued that the way to resolve semantic heterogeneity is though standard schemas. However, experience has shown that standards have very limited success and only in domains where the incentives to agree on standards are very strong. Even then, as the online retailer example illustrated, while data providers may share their data using a standard, their own data systems still employ their original schemas (and the cost of changing those systems is pro-

hibitive). Hence, semantic heterogeneity needs to be resolved at the step where the data provider exposes its data to its counterparts.

As one thinks about resolving schema heterogeneity, it is important to note several common instances of the problem, which may shed light on the specific problem at hand:

- One schema may be a new version of the other.
- The two schemas may be evolutions of the same original schema.
- We may know that we have many sources modeling the same aspects of the underlying domain (horizontal integration).
- We may have a set of sources that cover different domains but overlap at the seams (vertical integration).

4. THE STATE OF THE ART

Resolving schema heterogeneity is inherently a heuristic, human assisted process. Unless there are very strong constraints on how the two schemas you are reconciling are different from each other, one should not hope for a completely automated solution. The goal is to reduce the time it takes a human expert to create a mapping between a pair of schemas, and enable them to focus on the hardest and most ambiguous parts of the mapping. For example, the tools that enabled building the web site at www.everyclassified.com required that we be able to map the fields of the web form to our own schema in one minute, on average.[1]

As would be expected, people have tried building semi-automated schema matching systems by employing a variety of heuristics (see [13] for a survey). Below we review a few of these and their limitations. We note that the process of reconciling semantic heterogeneity typically involves two steps. In the first, called schema matching, we find correspondences between pairs (or larger sets) of elements of the two schemas that refer to the same concepts or objects in the real world. In the second phase, we build on these correspondences to create the actual schema mapping expressions. The Clio Project at IBM Almaden [15] is a prime example of work on building the mapping expressions.

The following classes of heuristics have been used for schema matching.

- **Schema element names:** element names (e.g., table and attribute names) carry some information about their intended semantics. Hence, by looking at the names, possibly stemming the words first we can obtain clues for the schema matcher. The challenges involved in using names are that the use of synonyms is very common as is the use of hypernyms (words that are specializations or generalizations). Furthermore, we often see that same word being used with different meanings (homonyms). In addition, we often see abbreviations and concatenations of words appearing in element names.
- **Data types:** schema elements that map to each other are likely to have compatible data types, but this is certainly not a rule. However, in many schemas the data types are underspecified (e.g., CDATA for XML). In practice, considering data types is a useful heuristic for ruling out certain match candidates.
- **Data instances:** elements from two schemas that match each other often have similar data values. Similarities can arise in several ways: (1) values drawn from the same small domain, e.g., makes of cars or names of countries, (2) significant occurrences of the same values (e.g., superlatives describing houses for sale), or (3) patterns of values (e.g., phone numbers, price ranges). Data instances are extremely useful when available, but one cannot rely on their availability.
- **Schema structure:** Matching elements in a schema are typically related to other related schema elements. For example, in an object-oriented hierarchy, it's often the case that if two classes match each other, then the children of these classes will also (at least partially) match. In XML DTDs proximity of attributes in a DTD (e.g., a Phone field next to Agent may suggest that it is the phone of the agent). However, relying on such a heuristic can be very brittle, and one of the main challenges is to find an initial match that drives the similarity of its neighbors.
- **Integrity constraints:** Considering integrity constraints on single attributes or across attributes can be useful for generating matches. For example, if two attributes are known to be keys in their respective schemas, then that provides additional evidence for their similarity.

While each of these heuristics is useful, experience has shown that taking any of them in isolation leads to a brittle schema matching solution. Hence, research has focused on building systems that combine multiple heuristics [2, 3, 12].

Despite these ideas, commercial products rely on completely manual specification of semantic mappings. They help by offering visual interfaces that enable designers to draw the lines between elements of disparate schemas, while the nitty gritty details of the mapping can often be generated on the back end. These tools already save a significant amount of time, but they do not suggest mappings to the designer.

5. AN EMERGING SOLUTION: LEVERAGING PAST EXPERIENCE

One of the fundamental reasons that the schema matching solutions described above are brittle is that they *only* exploit

[1]Mapping web forms also involves challenges beyond semantic heterogeneity, especially in the age of JavaScript.

evidence that is present in the two schemas being matched, ignoring past experience. These schemas often lack sufficient evidence to be able to discover matches. However, looking more closely at schema matching tasks, it is evident that these tasks are often repetitive. Specifically, we often find that we repeatedly map schemas in the same domain into a common mediated schema. For example, creating the engine at www.everyclassified.com involved mapping thousands of web forms in the same domain into a common schema, the one exposed to the users by the engine itself. A human expert, after seeing many schemas in a particular domain, is able to map schemas much faster because she has seen many variations on how concepts in the domain are represented in schemas.

The challenge, therefore, is to endow the schema matcher with the same capabilities: leverage past experience. For example, once the system has been given several mappings in the domain of used cars, it should be able to predict mappings for schemas it has not seen before. As it sees more schemas in a particular domain, its predictions should become more accurate, and it should be more robust in the presence of variations.

This idea was explored for the past few years in several academic research settings [3, 7, 9, 10, 11], and has recently been applied commercially for the first time by Transformic Inc., the creators of www.everyclassified.com.

The research projects considered the use of Machine Learning as a mechanism for enabling a schema matcher to leverage previous experience. In Machine Learning, the system is provided a set of *training examples*, and uses them to learn models of the domain of interest. In this context, the training examples are schema mappings that are manually constructed by domain experts and given to the system. The models of the domain enable the system to look at a new schema and predict a schema map. For example, the system can learn that the attribute concerning house descriptions typically involves a long text and include frequent occurrences of superlatives. Furthermore, the system can learn variations on the ways people name this field in practice.

Web-service search engine: Another application of this idea is search for web services, namely locating web services (or operations within them) that are relevant to a particular need. Simple keyword search does not suffice in this context because keywords (or parameter names) do not capture the underlying semantics of the web service. The Woogle Search Engine (described in a paper [4], and available at www.cs.washington.edu/woogle) is based on analyzing a large collection of web services and clustering parameter names into semantically meaningful concepts. These concepts are used to predict when two web service operations have similar functionality.

What can you learn from the past?

The paradigm of learning from past experience of performing schema matching tasks is only in its infancy. It is interesting to take a step back and consider what one can learn from the past in this context.

We assume the *past* is given to us as a collection of schemas in a particular domain, mappings between pairs of schemas in that collection, and to the extent possible, data instances. The schemas can come from anywhere, and can involve very closely related domains, not necessarily modeling the same data. In many cases, such schemas can be obtained from the Web or resources such as xml.org. In others, they may be available throughout an enterprise. We often refer to such a collection of schemas as a *corpus*, in analogy to the use of corpora of documents underlying Information Retrieval (IR) and web-search engines. Of course, while the corpora in IR involve collections of words, here we are managing semantically richer elements, such as schemas and their instances.

The goal of analyzing a corpus of schemas and mappings is to provide hints about deeper domain concepts and at a finer granularity. Looking a bit closer at the approach, the following are examples of what we can learn from a corpus.

- **Domain concepts and their representational variations:** As a first step, we can analyze a corpus to identify the main concepts in the domain. For example, in a corpus of book inventory schemas, we may identify the concept of book and warehouse and a cluster of price-related elements. Even more importantly, we will discover *variations* on how these concepts are represented. The variations may differ on naming of schema elements, grouping attributes into tables or the granularity of modeling a particular concept. Knowledge of these variations will be leveraged when we match two schemas in the domain.

- **Relationships between concepts:** Given a set of concepts, we can discover relationships between them, and the ways in which these relationships are manifested in the representation. For example, we can find that the Books table typically includes an ISBN column and a foreign key into an Availability table, but that ISBN never appears in a Warehouse table. These relationships are useful in order to prune candidate schema matches that appear less likely. They can also be used to build a system that provides advice in *designing* new schemas.

- **Domain constraints:** We can leverage a corpus to find integrity constraints on the domain and its representations. For example, we can observe that ISBN is a foreign key into multiple tables involving books, and hence possibly an identifier for books, or discover likely data types for certain fields (e.g., address, price). Constraints may have to do with *ordering* of attributes. For example, in a corpus of web forms about cars for sale, we may discover that the make attribute is always placed before the model and price attribute, but occurs after the new/used attribute.

Typically, constraints we discover in this way are *soft constraints*, in the sense that they are sometimes violated, but can still be taken as rules of thumb about the domain. Therefore, they are extremely useful in resolving ambiguous situations, such as selecting among several candidate schema matches.

6. LOOKING FORWARD

The need for flexible data sharing systems, within and across enterprises is only in its infancy. The tools we have today lag far behind customer needs. The problem is only exacerbated by the fact that much more of the data we need to manage is semi-structured, and is often the result of trying to *extract* structure from unstructured data. Hence, we need to manage data where the values, attributes names and semantics are often uncertain.

Going forward, I see two major challenge areas: dealing with drastically larger schemas and dealing with vastly more complex data sharing environments. In both of these, I think we may have to change the way we think about the problem at hand. I touch on each in turn.

Larger Schemas and Schema Search

The techniques described above deal with small to medium size schemas (including up to hundreds of elements). To their credit, it should be noted that the techniques gracefully handle large numbers of such schemas. It is well known that many real-world schemas have thousands of schema elements (tables and attributes), the SAP schemas being a prime example. The challenge of creating schema mappings is considerably harder here: you cannot even view the entire schema on a screen or multiple screens.

I believe there are two principles that need to guide the work on mapping larger schemas. The first is that the schema matching tools need to incorporate advanced information visualization methods (see [14] for an example of such work). Much of the challenge in designing schema mappings for large-scale schemas is ensuring that the attention of the designer is constantly directed to the right place, that they can view hypothetical mappings and remove them easily, and that they can see effectively how one fragment of the mapping may affect other fragments. The system should also be able to explain why certain match predictions are being made.

The second principle I would put forth requires changing the way we think of schema matching. Specifically, I am proposing a *schema search engine*. The engine contains a set of indexes on the elements of a particular schema (or set of schemas). The engine takes as input schema element (e.g., table name, attribute, XML tag), schema fragments, or combinations of schema fragments and data instances. The engine returns a ranked list of schema elements in the indexed schema that are candidate matches. The interface should be as simple as we see today in search engines. The justification for such a tool is that much of the work in schema mapping is simply finding where in a huge schema there are relevant fragments to the part of the schema that is currently under consideration, or alternatively, finding a relevant schema in a large collection of schemas. Hence, instead of focusing on tools that solve the entire problem, but are inherently *brittle*, build *robust* tools that bring the users closer to their needs. The Woogle web-service search engine previously described is an example of such an engine, but searching over web-service operations rather than schemas and their fragments.

Managing Dataspaces

A much greater challenge facing the data management community is to raise the abstraction level at which data is managed. Today, we have powerful systems for managing data at the level of a single database system (whether relational, XML, or in some other model). However, the data management challenges we face are at a much higher level: we need to manage a *dataspace*, rather than a database.

A dataspace is comprised of a set of *participants* and a set of *relationships*. Participants are individual data sources: relational databases, XML repositories, text databases, web services, data stream systems, sensor deployments, or any other element that stores or delivers data. Some participants may be *transparent*, with a full language for posing queries; a prime example is a traditional relational DBMS. Other participants may be *opaque* – offering limited interfaces for posing queries (usually supported by specific programs); examples are web services, stored procedures, and other software packages. In addition, some participants may have no structure to their data (e.g., text), or only some structure (e.g., code collections). Examples of dataspaces include: an enterprise, the desktop, a library, large scientific projects, a smart home, or a battlefield.

A dataspace should be able to model any kind of relationship between two (or more) participants. In the extreme case, a relationship is a full schema mapping that enables arbitrary data exchange and query reformulation among participants. In other cases, the relationship can express simple dependencies, where the details are not known precisely (e.g., one participant is an evolved version of another). The relationships can have a temporal aspect, e.g., how frequently data is exchanged, or that one is a mirror or backup of the other.

The key distinguishing feature of dataspace management is that integrations should evolve over time and as needed, but data should be accessible in some form from the very start. This means that simple queries (e.g., keyword queries) should always be supported on every participant in the dataspace without any effort. As the owners of the dataspace want to create more tight integration between sources and support more complex queries across participants, they can create more detailed semantic mappings as necessary. In addition, the management of dataspaces should consider the entire life cycle of data, including its acquisition, curation, query and update, evolution and analysis. The initial ideas on manag-

ing dataspaces are described in [5], but are only starting to intrigue the research community. Practitioners have so far embraced the idea with enthusiasm.

7. ACKNOWLEDGMENTS

The ideas espoused in this paper have benefited from many discussions and hard work by my colleagues and students. In particular, I'd like to thank Phil Bernstein, Anhai Doan, Luna Dong, Dana Florescu, Zack Ives, and Jayant Madhavan. The vision of dataspaces is the brainchild of discussions with Mike Franklin and Jennifer Widom.

8. AUTHOR BIOGRAPHY

Dr. Alon Halevy is a Professor of Computer Science at the University of Washington. He received his Ph.D in Computer Science from Stanford University in 1993 and then spent several years at AT&T Bell Labs. Dr. Halevy's research interests are in data integration, semantic heterogeneity, personal information management, management of XML data, web-site management, peer-data management systems, and the intersection between Database and AI technologies. His research developed several systems, such as the Information Manifold data integration system, the Strudel website management system, and the Tukwila XML data integration system. He was also a co-developer of XML-QL, which later contributed to the development of XQuery standard for querying XML data.

In 1999, Dr. Halevy co-founded Nimble Technology, one of the first companies in the Enterprise Information Integration space. In 2004, Dr. Halevy founded Transformic Inc., a company that creates search engines for the deep web, content residing in databases behind web forms.

Dr. Halevy was a Sloan Fellow (1999-2000), and received the Presidential Early Career Award for Scientists and Engineers (PECASE) in 2000. He serves on several journal editorial boards, and he served as the program chair for the ACM SIGMOD 2003 Conference. For additional information, see www.cs.washington.edu/homes/alon.

REFERENCES

[1] K. Aberer. Peer to peer data management: Introduction to a special issue. *SIGMOD Record*, 32(3), September 2003.

[2] H.-H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proc. of VLDB*, 2002.

[3] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: a machine learning approach. In *Proc. of SIGMOD*, 2001.

[4] X. L. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proc. of VLDB*, 2004.

[5] M. Franklin, A. Halevy, and J. Widom. Dataspaces: a new abstraction for data management, 2005.

[6] A. Halevy. Learning about data integration challenges from day one. *SIGMOD Record*, 32(3):16–17, September 2003.

[7] A. Halevy, O. Etzioni, A. Doan, Z. Ives, J. Madhavan, L. McDowell, and I. Tatarinov. Crossing the structure chasm. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR)*, 2003.

[8] A. Y. Halevy, N. Ashish, D. Bitton, M. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka. Enterprise information integration: Successes, challenges and controversies. In *Proceedings of the ACM SIGMOD Conference*, 2005.

[9] B. He and K. C.-C. Chang. Statistical schema integration across the deep web. In *Proc. of SIGMOD*, 2003.

[10] A. Hess and N. Kushmerick. Learning to Attach Semantic Metadata to Web Services. In *Proceedings of the International Semantic Web Conference*, 2003.

[11] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2005.

[12] J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.

[13] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[14] G. G. Robertson, M. P. Czerwinski, and J. E. Churchill. Visualization of mappings between schemas. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2005.

[15] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data Driven Understanding and Refinement of Schema Mappings. In *Proceedings of the ACM SIGMOD*, 2001.