Context-Sensitive Clinical Data Integration

James F. Terwilliger[1], Lois M. L. Delcambre[1], and Judith Logan[2]

[1] Department of Computer Science
Portland State University, Portland OR 97207, USA
{jterwill, lmd}@cs.pdx.edu,
[2] Department of Medical Informatics and Clinical Epidemiology
School of Medicine, Oregon Health and Science University, Portland OR 97239, USA
loganju@ohsu.edu

Abstract. Current methods for data integration are as difficult to use as they are powerful. Motivated by our work with clinical data and the people who analyze it, we present two components that allow non-technical users that are domain experts to create and reuse complex data integration processes. The GUAVA (GUI As View Apparatus) component enables data analysts to make informed data integration decisions based on detailed accounts of the user interface that was used to generate the data. The MultiClass component allows analysts to revisit decisions made for prior studies and reuse them or not each time the data is used. We describe these two components with examples where a warehouse of clinical data is used to support research studies. We describe the state of our implementation and why we believe the two components can be automatically translated into ETL workflows.

1 Introduction

In a traditional data warehouse, database specialists construct an ETL (Extract-Transform-Load) workflow to combine, and transform data from heterogeneous data sources and accumulate it in a central place. Because ETL can include programming code of any complexity, if a warehouse-building process can be done, it can be done using ETL. However, an ETL workflow, once defined, encapsulates only one set of decisions about how to integrate various source databases, and is almost entirely inaccessible to those who are not database experts or programmers. If health-related databases are to be used in research, data must be extracted from data sources and transformed differently for different studies, at multiple points in time, using methods to be specified by clinical experts rather than database experts.

In a clinical setting, there are additional weaknesses to the classical approach of full data integration. First, it may be necessary to lose information. A data source A with two categories, smokers or non-smokers, cannot be fully integrated with a data source B with three related categories, non-smokers, cigar smokers, or cigarette smokers without making a classification decision or declaring the integration impossible. Many integration techniques [1, 9, 15] identify similarities between data sources, without offering any guidance on how to answer the classification decision appropriately.

Second, integration techniques that do address the classification decision [6, 7] assume that there will be a single, integrated data source. We are interested in supporting an environment where data is used in different studies and may require different classifiers.

Because data integration solutions often require a person to read and understand a database, it is often left to technical experts to decide how to integrate data, even if they do not fully understand the data they are integrating. But, the data in the database is not sufficient for most clinical inquiries. The user interface of a software tool used to capture data defines the precise meaning of data. A "1" in the field "smoker" might mean that the patient is a current smoker, or instead could mean that they quit smoking one year ago.

We present two complementary components that allow domain experts to make their own integration and classification decisions, as needed, for each study. The first, GUI As View Apparatus (GUAVA), provides the user with a rich query interface that is derived from the same GUI that clinical providers used to assemble and view the data originally. Users can thus view data in its original context rather than the potentially obscure environment of a database. The second component, MultiClass, allows domain experts to integrate and classify data again and again, as needed. MultiClass captures these decisions and uses them to generate ordinary ETL workflows. Analysts are also able to use MultiClass to document, inspect, reuse, and modify integration decisions from prior studies.

Section 2 motivates this work by describing patient data used in clinical outcomes research. We describe the GUAVA and MultiClass components in Section 3. Section 4 looks at current results. Section 5 briefly presents related research. Section 6 describes our plans for future work and offers some conclusions.

2 The Status Quo

The primary actor in our scenario is the data analyst, a person trained in statistical methodology with good domain knowledge. For clinical studies, this means the analyst understands medical terminology and data. The data analysts that we are observing work for the *C*linical *O*utcomes *R*esearch *I*nitiative[3] (CORI), an organization that studies clinical data to improve the practice of clinical endoscopy. To encourage clinics to submit data, CORI developed a software reporting tool that clinics can use to document endoscopic procedures. Data from the CORI software tool is periodically sent for inclusion in the CORI warehouse. An ETL workflow performs schema transformation and data cleaning while moving the data into the warehouse. Analysts then identify and extract relevant reports for import into a statistical package for each study. Here are two studies that the analysts may run:

Study 1: *We would like to find out, of all patients undergoing upper GI endoscopy, how many (what proportion) had the indication of "Asthma-specific ENT/Pulmonary Reflux symptoms"? Of these, include only those with no history*

[3] More information about CORI can be found at http://www.cori.org.

of renal failure and with cardiopulmonary and abdominal examinations within normal limits. How many of these suffered the complication of transient hypoxia? Of these, how many required each of the following interventions: surgery, IV fluids, or oxygen administration?

Study 2: *Of all procedures on ex-smokers, how many had a complication of hypoxia?*

These research studies as performed by an analyst are more than just queries. A study comprises all of the decisions that a data analyst makes from the time a request arrives to when final statistical analyses are run, and those decisions can change over time. Those decisions are also based on the precise semantics of both the study and the data; if a study defines an ex-smoker to be someone who has quit in the last year, but the user interface indicates that an ex-smoker is anyone who has ever smoked, the data may not be appropriate to use in that study.

The technical demands of writing an ETL workflow are beyond the capabilities of the CORI analysts. It is left to the development team to write the ETL workflow, and as a consequence, the analysts do not completely understand the process by which data arrives in the warehouse. Thus, they also cannot modify that process as new research questions arise. To complicate matters, several commercial reporting tool vendors have expressed an interest in contributing data to CORI's clinical data warehouse. Each new vendor necessitates a new ETL workflow, potentially for each study.

3 Architectural Overview

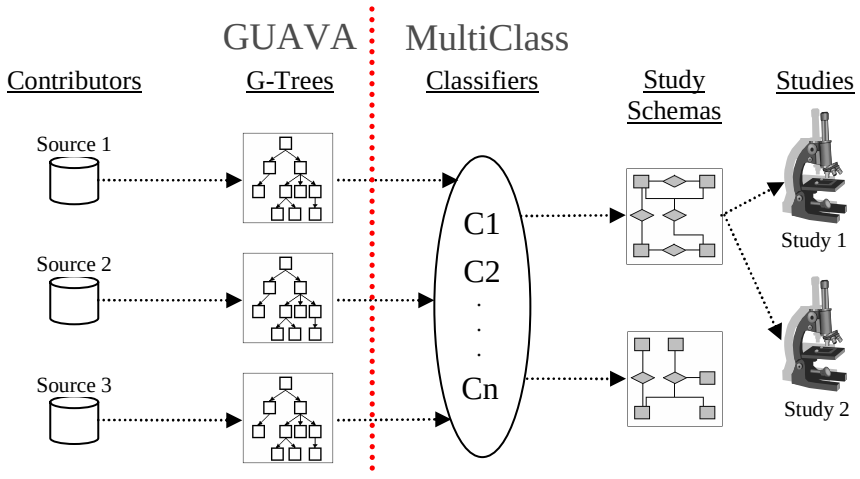


Fig. 1. GUAVA and MultiClass components and how they interface

The GUAVA and MultiClass components of our architecture enable data analysts to express their own data extraction, integration, and cleaning for each study. We introduce three artifacts (Figure 1):

- *GUAVA trees (g-trees)* that allow an analyst to explore the user interface of a data capture tool to select the data of interest
- *Study schemas* that document the data that analysts want in studies
- *Classifiers* that relate elements of g-trees with the study schema

Anyone using the system can annotate and timestamp each of these artifacts, as well as the studies themselves, so that it is clear who generated them, when, and why.

To perform a study, the data analyst chooses data elements from (or adds data elements to) a study schema, writes conditions similar to a WHERE clause in SQL to filter out unwanted data, and then selects or defines new classifiers. The analyst may choose to look at other studies that use the same study schema to make informed decisions as to which classifiers to use.

MultiClass uses the specifications set out by the analyst to create an ETL workflow that is tailored to a specific study. Thus, we can leverage existing ETL and still offer the flexibility that analysts require when running studies over semantically-rich data.

3.1 Scope

We do not expect our architecture to be a universal data integration solution. GUAVA expects that a GUI accompanies each data source. The user interfaces that interest us are *reporting tools*, where the primary purpose of the interface is to facilitate data entry.

We are not working on resolving naming conflicts or automated schema matching [1, 10]. We assume that, because data analysts are domain experts, they are capable of making judgments about domain-specific vocabulary, such as the fact that "interventions" in one source refers to the same data as "complications" in another source. Note that controlled vocabularies [4] or ontology, or other automated schema matching tools [10] may be useful in conjunction with GUAVA to assist the user.

Also, we are not addressing instance identification problem [15]. Since an endoscopy report is likely not created twice, MultiClass simply unions together the results of ETL workflows from different contributors.

3.2 GUAVA: GUI As View Apparatus

Each contributor schema in Figure 1 is associated with a GUAVA tree (or g-tree) that captures the structure and content of the user interface (Figure 2). The g-tree demonstrates relationships that may not be present in the database alone, such as a question that becomes enabled only if one answered a previous question in a specific way. Each node in a g-tree (Figure 3) captures context

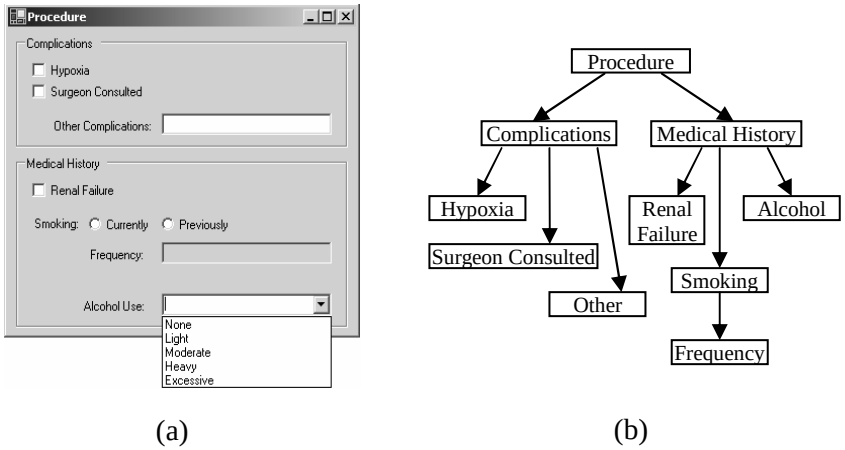


Fig. 2. An example dialog from a clinical tool and its corresponding g-tree. There is a node in the g-tree for every control on the screen, even those that do not normally store data, such as group boxes. Because the "frequency" textbox does not become enabled until someone answers the "smoking" question, the "frequency" node appears as a child of the "smoking" node

information about a control on the interface, including the exact wording of a control's question and answer options, whether there is a default value, and whether the control is required to be filled in.

The g-tree behaves like a view; when analysts write classifiers, they express queries against the g-trees. Thus, each node in the g-tree must refer back to the contributor's database to get data. As a normal part of using the reporting tool, when the user enters data into a field, the reporting tool places that data into the database. In GUAVA, we exploit that connection between UI elements and the database to generate mappings between the UI and the database automatically.

One benefit of our approach comes from how we deal with schematic heterogeneity, when information of interest appears as schema (such as table or column names) in one data source and as data (a field in a table extent) in another data source. The most frequent type of schematic heterogeneity arises because contributors often use a generic database layout, where each row in the database looks like "Entity, Attribute, Value". The user interface, however, is not generic and does not have a generic layout; if one considers a single screen in the interface as a row in the database, then each control represents a column.

Informally, we have noticed that reporting tools maintain an in-memory structure with a simple design: each screen of the tool corresponds to a table, and each control corresponds to a column. We call this design the *naïve schema* for a tool. The physical database design is far different, typically with a generic layout. We believe that the differences between the naïve schema and the real database can be encapsulated by specific design patterns (Table 1). Each

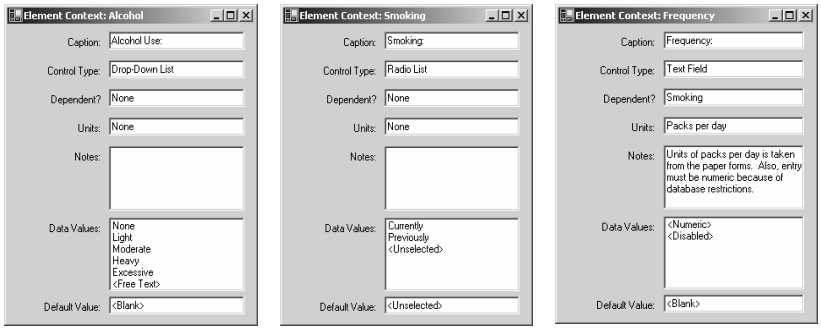


Fig. 3. Details for three nodes from the g-tree in Figure 2. The alcohol node (a) has one data value each for the selections in the drop-down list, and an option for free text. The smoking node (b) has an option for "unselected" because the radio list starts out with no option selected. The frequency node (c) records that the control does not become enabled unless the smoking control has an answer

pattern describes a data transformation; several put together describe how to translate a query against the g-tree into one against the database.

Since the code for a user interface can be arbitrarily complex, there may be an arbitrarily complex relationship between the UI and the underlying data source. By exploring the utility of database patterns, we hope to show that most such complex relationships can be expressed using a small number of design patterns.

3.3 MultiClass: Study Schema

A study schema collects all of the things that analysts want to study — like a patient's gender or smoking habits — and organizes them at a conceptual level. The study schema may be incomplete compared to a global schema. Data elements not needed in any study are simply omitted. Analysts can expand the study schema as needed for new studies. For the data analysts at CORI, the primary entity of interest is always the procedure; we expect that CORI would only need to have one study schema. We allow multiple study schemas, e.g., with patient or medications as the primary entity of interest.

A study schema simplifies the traditional ER model in that the only relationship type is "has-a" with a single entity of primary interest sitting atop a tree, much like a "part-of" hierarchy in a CAD database [13]. Using such a hierarchical model (Figure 4) meets the needs of clinical studies where the primary entity of interest is the procedure. The biggest difference between a study schema and an ER diagram is the addition of multiple *domains* for an attribute. Depending on the study, analysts may want to represent an attribute like smoking habits in different ways (Table 2).

Table 1. Example database design patterns. Each design pattern represents a transformation that one must perform when reading data into memory

Pattern	Description	Data Transformation
Naïve	No transformations are applied to the data.	None — this is just the in-memory database
Merge	Data from several forms are drawn from the same table.	Pull only data where C = form name (C is a column that holds forms)
Split	Attributes from a single form are distributed over several tables	Join
Generic	Each row in a table represents an attribute, rather than each column.	Execute an un-pivot operation, either in code or SQL if the operator exists in the DBMS
Audit	No rows are ever deleted or updated. Rows can be deprecated by setting the value in a column. The reporting tool only displays current data.	Pull only data where C = 0 (0 is a sentinel to indicate that the row has not been deleted)

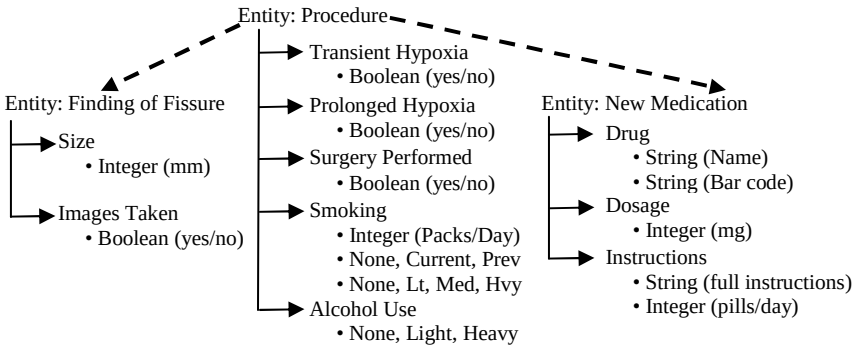


Fig. 4. A study schema. Entities have attributes, which in turn have domain(s) that correspond to different ways to represent them. The dashed lines indicate "has-a" relationships between entities, with the primary entity "Procedure" at the top of the tree

Table 2. Three different domains for the "smoking" attribute. There is no way to translate any one representation into another without losing information

Domain	Elements	Description
1	Positive Integers	Number of packs smoked per day
2	None, Current, Previous	No smoking, current smoker, or has smoked in the past
3	None, Light, Moderate, Heavy	General classification of smoking habits

3.4 MultiClass: Classifiers

An analyst creates a classifier to relate nodes in a g-tree with domain entries in a study schema. Each classifier is a list of declarative statements of the form $A \leftarrow B$, where A is an arithmetic calculation and B is a Boolean condition. Both clauses use nodes in a g-tree as arguments (see Figure 5 for examples). Thus, the input to a classifier is contributor data, but as displayed as it appears in a user interface rather than as stored in a database.

MultiClass allows more than one classifier to map data from the same contributor to the same domain. Different studies may interpret domain values differently; a "previous" smoker may mean someone who has quit in the last year, or in the last ten years, or at any time at all. MultiClass needs *entity classifiers* to identify unique objects in a g-tree and bring them forward into a study schema. An analyst creates an entity classifier just like any other classifier, except the target object of the classifier is an entity rather than a domain. Also, the classifier must refer to at least one node in the g-tree that represents a form rather than an attribute.

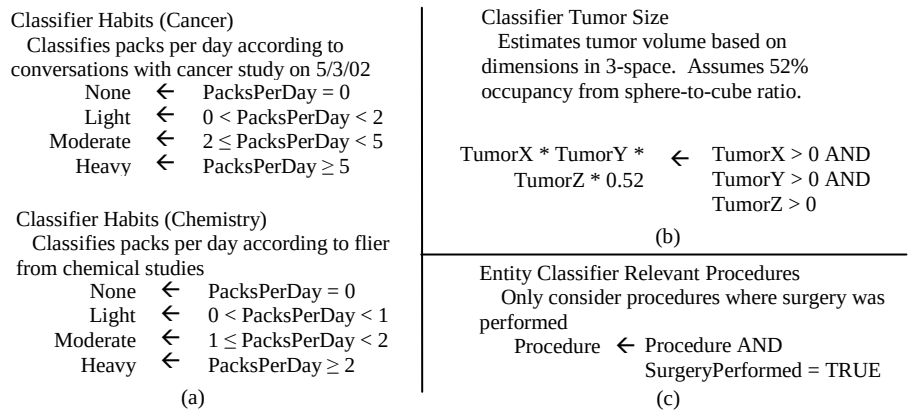


Fig. 5. Example classifiers. Two classifiers (a) can relate data from a contributor to the same domain for different studies. Another classifier (b) shows how to write classifiers that refer to more than one g-tree node. An entity classifier (c) tells MultiClass how to relate forms in the application with entities in the study schema, where "Procedure" is a node in the g-tree that represents the form in Figure 2

4 Analysis

We have shown a number of analysts our architecture with examples of g-trees, classifiers, and study schemas and compared them with the statistical software tools that they currently use. They confirm that g-trees are easy to read and that classifiers are simple to write and organize.

4.1 Research Directions

In the process of our research, we will investigate the validity of three hypotheses.

Hypothesis #1: It is possible to automatically generate a g-tree and database mappings using an Integrated Development Environment (IDE). The prototype of GUAVA that we are developing extends Visual Studio .Net to generate a g-tree from the code that makes up the GUI of a reporting tool. The prototype allows the developer to specify database design patterns that relate the g-tree to the database.

Hypothesis #2: The artifacts of GUAVA and MultiClass are simple enough that data analysts can use them without technical assistance. We assert that g-trees are significantly simpler to read than database schemas, that classifiers are easy to specify, and that domains are simple to understand because they are a concept from statistics. Usability testing will include measuring precision and recall; analysts should be able to extract only and all relevant data from contributors without technical help.

Hypothesis #3: It is possible to compile studies into ETL workflows. A study in MultiClass consists of classifiers that draw from databases using GUAVA. At this early stage, we can show how to translate these objects into an ETL workflow in specific cases. We aim to show that we can generate ETL workflows by comparing the expressive power of our classifier language against a set of common ETL components.

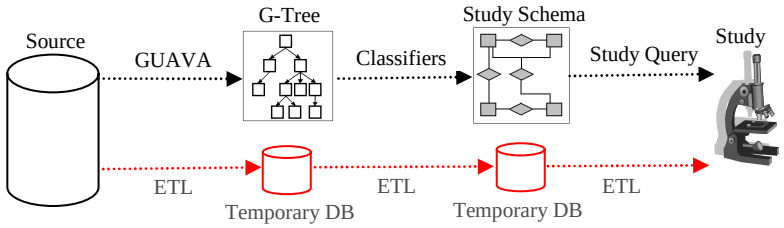


Fig. 6. Translating GUAVA and MultiClass artifacts into ETL

We believe that the classifier language as specified here is equivalent in expressive power to conjunctive queries with union. We can translate queries specified against the g-tree into predefined SQL queries and ETL components that depend on the database patterns used. At present, a study created over GUAVA and MultiClass has a logical translation to a sequence of three ETL components, each executing a query over the previous one's results (Figure 6).

4.2 Implementation Status and Options

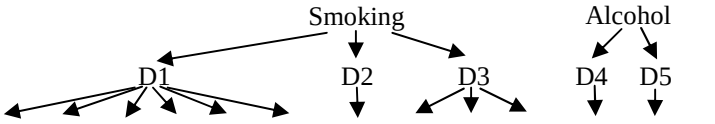
A prototype implementation of GUAVA is underway. Though we have identified 11 distinct database patterns so far, our initial prototype only considers the patterns listed in Table 1. The prototype takes the standard .Net form components

and extends them with methods that allow the IDE to generate a g-tree. The g-tree is stored as an XML Schema, which mimics the hierarchical nature of the form interface and allows queries to return XML documents in a standard format. Currently, we are working on developing the mechanism to translate queries against the tree.

We are still in the early stages of development for MultiClass. We have developed the algorithms for translating the classifier language into XQuery. Our approach is to identify all of the nodes in a g-tree that are referenced by the set of classifiers. Then, treat each entity classifier as a "for-each" to iterate through objects, each domain classifier as a variable assignment, and each rule in a classifier as a conditional statement. To date, we have successfully hand-translated several collections of classifiers into both XQuery and Datalog.

We are still considering our options for implementing a study schema. The naïve approach is to materialize the output of individual classifiers into relational tables or XML documents. In the relational model, the result is a collection of tables, one table per entity classifier per entity, with columns representing classifier output (Figure 7). This option allows for simple data retrieval because getting data from the study schema reduces to select-project-join queries. If the classifiers/domains ratio is high, then a comprehensive materialized study schema may be too large to manage. Alternatives include materializing only often-used classifiers or determining relationships between classifiers. The latter implies that if classifier A and classifier B share a simple algebraic relationship, then we can materialize A's output and compute B as needed.

Entity: Procedure, Data Source: CORI, Entity Classifier: Colonoscopies_Only



C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC
T	T	T	T	T	F	0	A	A	A	T	H
T	T	T	T	F	F	1	B	B	B	T	M
T	F	F	F	T	F	2	A	A	B	T	L
F	F	F	F	F	F	0	C	B	C	F	N

Classifiers

Fig. 7. A fully-materialized study schema must also materialize every classifier, where each classifier serves as a column in the table

5 Related Work

SEMEX. The SEMantic EXplorer project [2] also supports on-demand data integration by non-technical users. SEMEX uses outside sources such as search

engines to suggest matches, but does not consult any user interface that may have generated data. Also, SEMEX does not provide any way to classify values.
Schematic Heterogeneity. SchemaSQL [8] and nD-SQL [5] demonstrate how to extend SQL to accommodate schematic heterogeneity; however, few of these features are available in commercial databases. They are also very difficult to learn, even for expert SQL users, so even if we decide to use these languages in our implementation, we will not require data analysts to learn them.
Mediated Schemas. A mediated schema serves a similar function as a study schema: presenting a unified view of heterogeneous data sources to the end user — in our case, a data analyst. A mediated schema uses Inter-schema Correspondence Assertions (ICAs) to establish relationships between databases. Database patterns and classifiers both act as ICAs, if one were to consider g-trees to be schemas. There exist both a simple notation [12] and more complex notations such as GLAV [7] for representing ICAs. These notations only express relationships between sets of objects, such as equality or containment. They do not express any transformation of data elements as is required for classifiers.
Context. The COIN project [11] stores context alongside data in the form of metadata, and also automatically transforms data based on that context. Because a g-tree resides outside of the underlying database, GUAVA can attach context information over an existing database regardless of implementation. MultiClass allows for multiple classifiers to transform data from the same context to the same domain in different ways according to need.

6 Future Work and Conclusion

We want to extend the classifier language to allow data cleaning, since analysts may also choose to discard data based on the needs of the particular study they wish to run. We are also interested in handling new versions of a reporting tool by propagating classifiers to the next version if their input nodes did not change, and suggest new classifiers if there is a change. Finally, we are interested in exploring whether GUAVA or MultiClass is able to provide benefits in other domains, such as traffic data and financial applications.

7 Acknowledgements

This work is supported in part by Collins Medical Trust, and by DHHS NIH National Institute of Diabetes Digestive and Kidney Diseases No. 5-R33-DK061778-03 awarded to Oregon Health & Science University (OHSU).

References

1. R. Dhamankar, Y. Lee, A. Doan, A. Halevy, P. Domingos. iMAP: discovering complex semantic matches between database schemas. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, Paris, France, 2004, 383–394.

2. X. Dong, A. Y. Halevy. A Platform for Personal Information Management and Integration. In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA, USA, January 4-7, 2005, 119–130.
3. F. Du, S. Amir-Yahia, J. Freire. A comprehensive solution to the XML-to-relational mapping problem. In *Proceedings of the 6th Annual ACM International Workshop on Web Information and Data Management*, Washington DC, November 12-13, 2004, 31–38.
4. M. Evens. Thesaural Relations in Information Retrieval. In R. Green, C. A. Bean, and S. H. Myaeng (eds.), *The Semantics of Relationships: An Interdisciplinary Perspective*, Kluwer Academic Publishers, Dordrecht, Netherlands, ©2002, 143–160.
5. F. Gingras and L. V. S. Lakshmanan. nD-SQL: A multi-dimensional language for interoperability and OLAP. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB)*, New York City, USA, 1998, 134–145.
6. J. A. Larson, S. B. Navathe, and R. Elmasri. A Theory of Attribute Equivalence in Databases with Application to Schema Integration. *IEEE Transactions on Software Engineering*, April 1989, 15(4):449–463.
7. J. Madhavan and A. Y. Halevy. Composing Mappings Among Data Sources. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, Berlin, Germany, September 2003, 572–583.
8. R. J. Miller. Using Schematically Heterogeneous Structures. In *Proceedings of ACM SIGMOD*, Seattle, WA, June 1998, 27(2):189–200.
9. R. J. Miller, M. A. Hernandez, L. M. Haas, L.-L. Yan, C. T. H. Ho, R. Fagin, and L. Popa. The Clio Project: Managing Heterogeneity. *SIGMOD Record*, 2001, 30(1):78–83.
10. E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. In *Proceedings of the 27th International Conferences on Very Large Databases*, 2001, 10(4):334–350.
11. E. Sciore, M. Siegel, and A. Rosenthal. Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Transactions on Database Systems*, June 1994, 19(2):254–290.
12. S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogeneous schemas. *VLDB Journal*, 1992, 1:81–126.
13. D. L. Spooner. Towards an Object-Oriented Data Model for a Mechanical CAD Database System. In K. R. Dittrich, U. Dayal, and A. P. Buchmann (eds.), *On Object-Oriented Database Systems*, Springer-Verlag, Berlin, Germany, 1991, 189–205.
14. P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of ETL scenarios. *Information Systems*, November 2005, 30(7):492–525.
15. Y. R. Wang and S. E. Madnick. The inter-database instance identification problem in integrating autonomous systems. In *Proceedings of the Fifth International Conference on Data Engineering (ICDE)*, Los Angeles, CA, February 6-10, 1989, IEEE Computer Society Press, Washington, DC, pages 46–55.