Semi-Automatic Schema Integration in Clio*

Laura Chiticariu† Mauricio A. Hernández Phokion G. Kolaitis‡ Lucian Popa
UC Santa Cruz IBM Almaden Research Center
laura at cs.ucsc.edu {mauricio,kolaitis,lucian} at almaden.ibm.com

1. INTRODUCTION

Schema integration is the problem of finding a unified representation, called the integrated schema, from a set of source schemas that are related to each other. The relationships between the source schemas can be represented via correspondences between schema elements or via some other forms of schema mappings such as constraints or views. The integrated schema can be viewed as a means for dealing with the heterogeneity in the source schemas, by providing a standard representation of the data. Schema integration has received much of attention in the research literature [1, 2, 6, 8, 10] and still remains a challenge in practice. Existing approaches require substantial amount of human feedback during the integration process and moreover, the outcome of these approaches is a *single* integrated schema. In general, however, there can be *multiple* possible schemas that integrate data in different ways and each may be valuable in a given scenario.

In this demonstration we showcase a novel schema integration method that is capable of enumerating multiple interesting integrated schemas. In addition to the systematic enumeration of schemas, our system provides easy-to-use capabilities for searching and refining the enumerated schemas via user interaction, and a visual interface that facilitates the user's understanding of the schema integration task at hand.

The schema integration system is built in conjunction with Clio [7], a schema mapping system that produces a declarative mapping specification describing the relationship between a source and a target schema, based on a set of correspondences between their attributes. From the declarative mapping specification, Clio further generates executables (queries or code) that transform an instance of the source schema into an instance of the target schema. In particular, when used together with the schema integration module, Clio will generate the data transformations from the source schemas to the newly constructed integrated target schema.

*This work was funded in part by the U.S. Air Force Office for Scientific Research under contract FA9550-07-1-0223. †Supported in part by NSF CAREER Award IIS-0347065 and NSF grant IIS-0430994. Work partially done while at IBM Almaden. ‡On leave from UC Santa Cruz.

VLDB '07, September 23-28, 2007, Vienna, Austria.

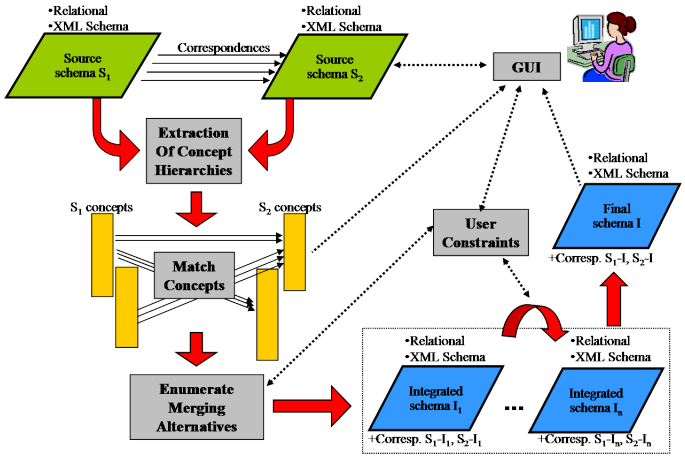

Figure 1: System architecture.

1.1 Schema Integration Process

Figure 1 depicts the architecture of our schema integration system. We now describe the schema integration procedure implemented in our system. Details of the algorithms and their performance can be found in the full version of this paper [3]. Section 2 describes this process using an example.

We assume we are given an integration scenario consisting of two or more relational and XML source schemas (describing data in a common domain) together with a set of *correspondences* that relate pairs of elements in these schemas. Correspondences signify "semantically equivalent" elements in two schemas. They can be specified by the user directly in the visual interface of Clio, or Clio may also infer correspondences using schema matching techniques [9]. For simplicity, we show only two source schemas in Figure 1. The architecture and the usage of the system remains the same in the more general case when more source schemas are used.

Concept Extraction: The first step of our integration method is to recast each source schema, with its constraints and nesting, into a higher-level concept hierarchy (with no constraints or nesting). Intuitively, a concept represents one category of data that can exist according to a schema (e.g., "project", "department", "employee of a department"). The concepts in a schema form a natural hierarchy (e.g., "employee of a department" is a sub-concept of "department"). To generate the concepts, we make use of an existing Clio module that extracts concepts from the relational or XML schemas; this module was used as a stepping stone towards the generation of mappings and queries that convert data from one schema to another [7]. In the same spirit, extraction of source concepts is a preliminary step

here towards generating the integrated schemas. Most of our subsequent integration method operates at the higher level of abstraction that is offered by the concept hierarchies.

Concept matching and merging: We identify *matching* concepts in different hierarchies, based on correspondences between their attributes. For every pair of matching concepts we have two alternatives: either merge them into one integrated concept, or leave them as separate concepts. We enumerate all possible ways of merging, to create an enumeration of the possible sets of integrated concepts. Each set of integrated concepts has its own hierarchy that is derived from the input concept hierarchies. Furthermore, each such integrated hierarchy of concepts can be recast (back) as an integrated schema. Hence, the result of this step is an enumeration of candidate integrated schemas.

For each integrated schema, as a byproduct of the generation process, the system also produces the set of correspondences between the elements of the source schemas and the elements of the integrated schema. Based on these correspondences, we can then use Clio's mapping generation component (not shown in Figure 1) to obtain a more precise (executable) specification of the relationship between the source schemas and the target.

Integrated Schema Refinement: The final step of our integration methodology is to interactively assist the user in deciding the final target schema. Users can browse and search through the generated schemas based on several criteria. Furthermore, we allow users to enter *constraints* on the merging process itself, based on their domain knowledge, as well as the integrated schemas they have already seen. For instance, users may specify that two or more source concepts should never appear merged in the integrated schema. We avoid the full enumeration of a potentially large space of integrated schemas by combining partial enumeration of schemas with the user constraints. We demonstrate how this user interaction can easily, and in a systematic way, narrow down the (potentially large) set of candidate schemas to a few relevant ones.

1.2 Technical Contribution

Our main technical contribution is the implementation of a schema integration system operating as outlined above. To the best of our knowledge, this is the first semi-automatic schema integration method that systematically enumerates multiple, meaningful, integrated schemas, while at the same time incorporating user feedback into the enumeration process. Our method applies to integration scenarios consisting of two or more relational and XML schemas, but operates at a logical, conceptual, level that abstracts away the physical details of relational or XML schemas and makes it easy to express user requirements. Hence, our method can be applied to any logical models (not necessarily extracted from schemas), for as long as they can be expressed as concept hierarchies. In addition, we have shown in [3] that our methodology has the following properties.

- Each generated integrated schema is capable of representing all information from the sources and does not represent any extra information not present in the sources. Moreover, the structure present in the source schemas is preserved, as much as possible.
- Our enumeration algorithm avoids duplication of work during the exploration of the space of merging configurations by avoiding to enumerate different configurations that yield the same schema. In doing so, we make use of a *polynomial-delay* algorithm [4] for enumeration of all boolean vectors satisfying a set of Horn clauses. Polynomial-delay [5] means that the delay between generating any two consecutive outputs (satisfying assignments in our case) is bounded by a polynomial in the size of the input. Intuitively, this is the best one can do when the number of outputs is exponential in the size of the input.

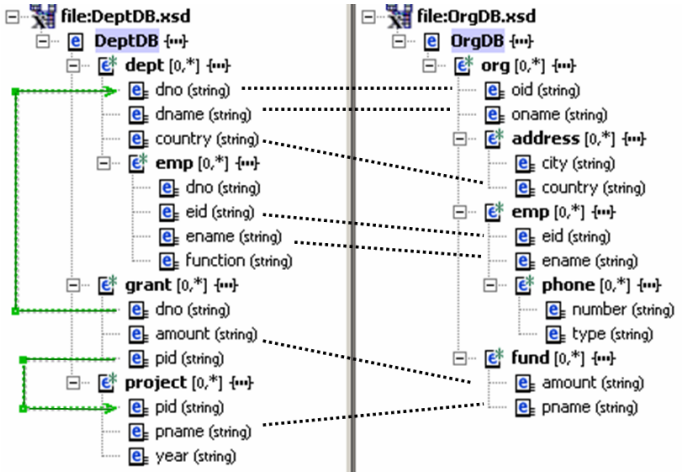

Figure 2: Two source schemas and their correspondences.

2. DEMONSTRATION OVERVIEW

In this section we describe our methodology by means of a simple example integration scenario shown in Figure 2. We emphasize that schema integration tasks often deal with large source schemas. During the actual demonstration, we also plan to use a real life and larger scenario to show how our techniques facilitate the task of schema integration and scale in such scenarios. We will also illustrate n-way schema integration scenarios (with $n > 2$) showing how multiple related schemas can be merged into one schema.

2.1 Example Integration Scenario

Figure 2 shows two input XML schemas: DeptDB and OrgDB. DeptDB represents information about departments with their employees and grants, as well as the projects for which grants are awarded. The arrows in DeptDB represent referential constraints: a grant has references to both a department and a project elements. OrgDB is a variation of DeptDB, where employees and funds are now grouped by organizations. Furthermore, OrgDB includes additional information not present in DeptDB (e.g., organizations have addresses and employees have phones.)

Given such source schemas, the preliminary step towards the computation of an integrated schema in our method is to provide a set of *correspondences* between the schemas. These correspondences are bi-directional and signify potentially equivalent elements in the two schemas. We only need to consider correspondences between atomic elements, which are the elements that carry actual data. We also use the name *attributes* for such atomic elements. In our implementation, correspondences are manually entered (drawn) by the user using Clio. Figure 2 depicts the correspondences as the lines across the two schemas. Not all attributes need to be an end-point of a correspondence. Also, in general, an attribute can be the end-point of multiple correspondences.

2.2 Extracting Concept Hierarchies

Once mappings between source schemas are established, users can request the creation of integrated schemas. The first step of our integration methodology is to convert each source schema into the corresponding concept hierarchy. The concept hierarchies that are implicit in our example source schemas are illustrated in Figure 3A. A concept is specified as a relation name with a set of associated attributes. For example, the concepts of a "department" and of an "employee of a department" that are implicit in the schema DeptDB are represented as the relations dept and respectively, emp with the

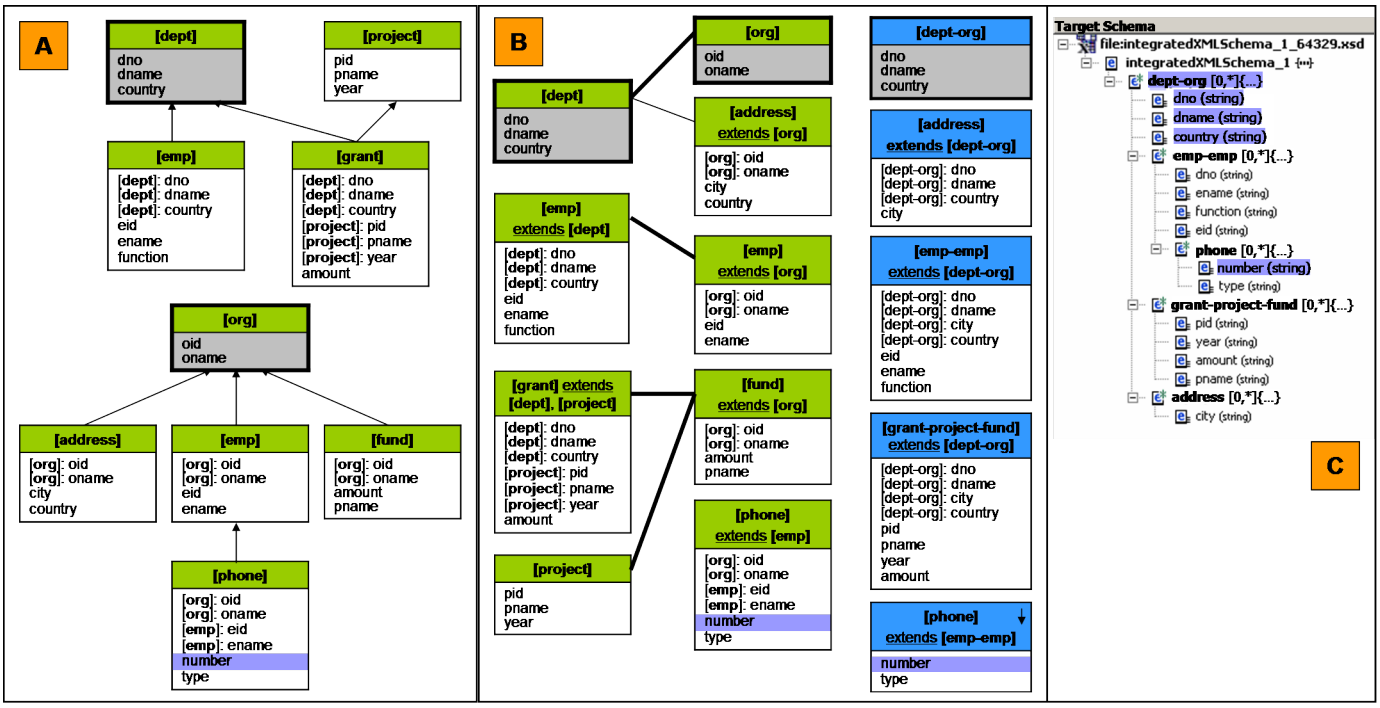

Figure 3: Screenshots from the visual interface: (A) source concepts hierarchies; (B) graph of matching source concepts and an integrated hierarchy; (C) an integrated schema.

corresponding attributes shown in Figure 3A.

Some concepts, such as dept and project, are top-level concepts and correspond to the top-level sets of a schema. These are standalone concepts; for example, a department and, respectively, a project tuple may exist in an instance of DeptDB independently of any other elements. In contrast, emp is a *sub-concept* of dept, since an employee element can only exist in the context of a department, according to the nesting structure of DeptDB. Consequently, the emp concept includes all attributes of an employee element together with the attributes in department. Similarly, grant is a sub-concept of both dept and project, due to the referential constraints in DeptDB, and inherits all attributes of these concepts.

In our visual interface, the sub-concept relationship is shown via directed edges from a concept to its immediate super-concepts. (To avoid clutter, we do not show transitive edges in the hierarchies.) Also, we prefix all inherited attributes with the name of the super-concept that they are inherited from, to distinguish them from attributes *specific* to that concept (i.e., that do not appear in any of its super-concepts). For example, the attributes inherited from department are prefixed with dept: in the employee concept.

2.3 Enumerating Integrated Schemas

The second step of our approach is to merge the concept hierarchies in all possible ways. In doing so, we first identify *matching* concepts in different hierarchies by taking into account the correspondences between their attributes.

Two concepts A and B match (i.e., they can be merged) if and only if there is a correspondence between an attribute specific to A and an attribute specific to B. For example, dept matches with org and hence they may be merged. However, grant and address do not match, since there is nothing specific to grant that matches something specific to address. Indeed, it is unintuitive to merge grant with address just because their attributes inherited from dept and respectively, org match. However, it is intuitive to merge grant (or project) with fund.

Figure 3B illustrates a different view of the concept hierarchies. (Ignore the third column of concepts for now.) This view illustrates the *graph of matching concepts*, where every pair of concepts that match is connected by a *matching* edge. The sub-concept edges are not shown in this view, to avoid clutter. Rather, they are specified through simple class extension notation. For example, we write emp extends dept to specify that emp is a direct sub-concept of dept.

A set of integrated concepts is obtained by merging source concepts along the matching edges in a given configuration. A configuration simply means selecting a *subset* of the matching edges. Our system considers all possible schemas that can result by selecting different such subsets. The third column in Figure 3B shows the integrated concepts obtained given a configuration consisting of all the matching edges in our example, except the one between dept and address (notice that the edge is not highlighted in the figure). According to this configuration, we merge dept with org, grant with fund and project, and also the two emp concepts. For example, the integrated concept denoted as dept-org in the figure is obtained by merging dept with org. (The names of integrated concepts are computed by concatenating the names of "contributing" source concepts, and they can be further customized by a user.)

As another example, consider the configuration consisting of all the matching edges in the graph. This will result in a different set of integrated concepts, where address is merged with dept and org.

In [3] we detail a two-step procedure that given a merging configuration, constructs a hierarchy of integrated concepts while preserving the hierarchies of source concepts. The first step is to create an integrated concept for every group of input concepts that are to be merged. The attributes of the integrated concept are obtained by unioning together the attributes of contributing concepts and elimi-

nating duplicates. Intuitively, two attributes a and b are *equivalent* (thus duplicates) if they are the same attribute in a source schema or they are connected by a correspondence, or there is an attribute c that is equivalent to both a and b. For example, the attributes dno and oid are equivalent. Hence, only one of them appears in dept-org (e.g., dno). In the second step, we construct a hierarchy on top of the set of integrated concepts, based on the source concept hierarchies. For example, we set address to be a sub-concept of dept-org, since address is a sub-concept of org in the second source schema and org contributes to dept-org.

The hierarchy on the integrated concepts is used to construct an actual schema. Figure 3C shows the nested schema that we generate from the hierarchy of integrated concepts shown in Figure 3B. Intuitively, each integrated concept corresponds to a set-valued element in the integrated schema. Moreover, sub-concepts are nested inside one of their direct super-concepts (without repeating the inherited attributes). We use foreign keys (keyref in XML Schema) to encode the rest of the super-concepts (when there is more than one). When computing the attributes in the integrated concepts, we also remember where each attribute is created from, in order to create correspondences between attributes of the source schemas and attributes of the integrated schema. For example, two correspondences are created for the attribute dept-org/dno in the nested schema of Figure 3C, corresponding to DeptDB/dept/dno and OrgDB/org/oid.

An important property of our integration procedure is that the integrated schemas that we generate preserve the structures of the source schemas (modulo merging of the matching concepts). It is easy to observe vestiges of the two source schemas in Figure 3C. For example, employees are still nested under organizations, as in OrgDB. Similarly, fund elements are still nested under organizations, except that their structure is now enriched with information about projects coming from DeptDB.

A naive enumeration algorithm would consider enumerating all subsets of matching edges (i.e., all configurations) and computing a set of integrated concepts (and corresponding schema) in each case. However, distinct configurations may result in identical sets of integrated concepts (and identical schemas). We have implemented a more efficient strategy that avoids a large percentage of the configurations that are guaranteed to result in duplicate schemas. The resulting algorithm has significant performance benefits over the naive enumeration and makes use of a polynomial-delay algorithm for generating all satisfying assignments for a set of Horn clauses [4]. We refer the interested reader to [3] for the details.

2.4 User Constraints

The third step of our integration technique is to interactively assist the user in deciding the final integrated schema. Our enumeration procedure outputs integrated schemas one by one and the user is allowed to browse the schemas generated so far, as well as request the generation of a new schema. The user may also add constraints at any point in the enumeration procedure. Once a new constraint $\mathcal{F}$ is specified, the generated schemas that do not satisfy $\mathcal{F}$ are filtered out. Furthermore, if the enumeration is not finished, we continue enumerating only the schemas that satisfy $\mathcal{F}$, in addition to the rest of the constraints.

User constraints are given using our visual interface in terms of the concepts and of the matchings between them. For example, the user can enforce a pair of matching concepts to be always merged or never merged (e.g., "merge dept with org", or "do not merge dept with address"). Additionally, the user can give constraints that enforce the preservation of certain structural patterns in the source schemas. For example, a constraint that specifies to not merge org, address, emp, phone, fund requires the structure of the second schema OrgDB to be preserved. However, the concepts of the first schema can be freely merged into the concepts of the second schema, based on the matching edges.

2.5 Visual Interface

The visual interface plays an essential role in facilitating the user's understanding of the integration scenario. The interface displays multiple, correlated views of the integration scenario:

1. The source schemas and correspondences (Figure 2).
2. The source concept hierarchies (Figure 3A).
3. A side-by-side view of the graph of matchings and the integrated concepts currently being analyzed (Figure 3B).
4. The corresponding integrated schema (Figure 3C).
5. The correspondences between the source and integrated schemas (not shown in the figure).

All views are synchronized in such a way that related entities across multiple views are highlighted when selected on any one view. For example, selecting the dept-org integrated concept in Figure 3B has several effects. First, the contributing source concepts, together with the corresponding matching edges are highlighted (Figures 3A-B). Sets in the source schemas that "contribute" attributes to these concepts (not shown in the figure), and the set in the integrated schema that corresponds to dept-org (Figure 3C) are also highlighted. Similarly, selecting an attribute allows a user to visualize all "related" attributes in the schemas and concept hierarchies (e.g., see number in Figures 3A-C).

To facilitate the visualization of matching concepts, matching concepts and their corresponding integrated concept are aligned to appear closer to each other (as depicted in Figure 3B). Furthermore, the appearance of various elements is customizable. The user can resize, minimize or maximize concepts, hide inherited attributes (see phone in Figure 3B) or collapse entire groups of concepts in order to eliminate clutter and be able to concentrate on specific parts of the integration problem.

3. REFERENCES

[1] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[2] P. Buneman, S. B. Davidson, and A. Kosky. Theoretical Aspects of Schema Merging. In *EDBT*, pages 152–167, 1992.

[3] L. Chiticariu, P. Kolaitis, and L. Popa. Semi-Automatic Generation and Exploration of Schema Integration Alternatives. Manuscript under submission, 2007.

[4] N. Creignou and J. Hébrard. On generating all solutions of generalized satisfiability problems. *ITA*, 31(6):499–511, 1997.

[5] D. S. Johnson and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[6] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The Use of Information Capacity in Schema Integration and Translation. In *VLDB*, pages 120–133, 1993.

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

[8] R. Pottinger and P. A. Bernstein. Merging Models Based on Given Correspondences. In *VLDB*, pages 826–873, 2003.

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

[10] S. Spaccapietra and C. Parent. View Integration: A Step Forward in Solving Structural Conflicts. *IEEE Trans. Knowl. Data Eng.*, 6(2):258–274, 1994.