SEEKing Knowledge in Legacy Information Systems to Support Interoperability

Joachim Hammer, Mark Schmalz, William O'Brien[¥], **Sangeetha Shekar** and **Nikhil Haldevnekar**
Dept. of Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32605, U.S.A.

Abstract. The SEEK project (Scalable Extraction of Enterprise Knowledge) is developing methodologies to overcome the problems of assembling knowledge resident in numerous legacy information systems by enabling rapid connection to, and privacy-constrained filtering of, legacy data and applications with little programmatic setup. In this report we outline our use of data reverse engineering and code analysis techniques to automatically infer as much as possible the schema and semantics of a legacy information system. We illustrate the approach using an example from our construction supply chain testbed.

1 MOTIVATION

We are developing methodologies and algorithms to facilitate discovery and extraction of enterprise knowledge from legacy sources. These capabilities are being implemented in a toolkit called SEEK (Scalable Extraction of Enterprise Knowledge). SEEK is being developed as part of a larger, multi-disciplinary research project to develop theory and methodologies in support of computerized decision and negotiation support across a network of firms (general overview in [6]). SEEK is not meant as a replacement for wrapper or mediator development toolkits. Rather, it complements existing tools by providing input about the contents and structure of the legacy source that has so far been supplied manually by domain experts. This streamlines the process and makes wrapper development scalable.

Figure 1 illustrates the need for knowledge extraction tools in support of wrapper development in the context of a supply chain. There are many firms (principally, subcontractors and suppliers), and each firm contains legacy data used to manage internal processes. This data is also useful as input to a project level decision support tool. However, the large number of firms working on a project makes it likely that there will be a high degree of physical and semantic heterogeneity in their legacy systems. This implies practical difficulties in connecting firms' data and systems with enterprise-level decision support tools. It is the role of the SEEK toolkit to help establish the necessary connections with minimal burden on the underlying firms, which often have limited technical expertise. The SEEK wrappers shown in Fig. 1 are wholly owned by the firm they are accessing and hence provide a safety layer between the source and end user. Security can be further enhanced by deploying the wrappers in a secure hosting infrastructure at an ISP, for example, as shown in the figure.

We note that SEEK is not intended to be a general-purpose data extraction tool: SEEK extracts a narrow range of data and knowledge from heterogeneous sources. Current instantiations of SEEK are designed to extract the limited range of information needed by these process models to support project optimization.

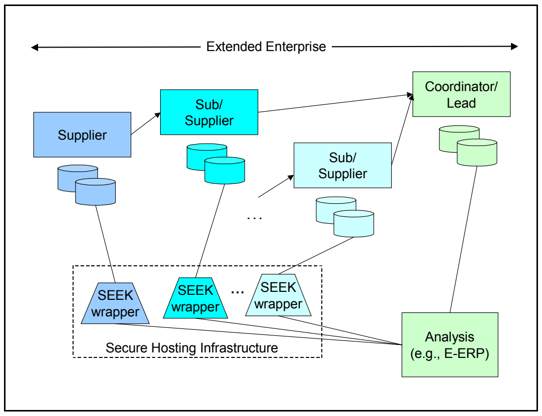

Figure 1: Using the SEEK toolkit to improve coordination in extended enterprises.

2 SEEK APPROACH TO KNOWLEDGE EXTRATCION

SEEK applies Data Reverse Engineering (DRE) and Schema Matching (SM) processes to legacy database(s), to produce a source wrapper for a legacy source. The source wrapper will be used by another component (for the analysis component in Figure 1) wishing to communicate and exchange information with the legacy system.

First SEEK generates a detailed description of the legacy source, including entities, relationships, application-specific meanings of the entities and relationships, business rules, data formatting and reporting constraints, etc. We collectively refer to this information as *enterprise knowledge*. The extracted enterprise knowledge forms a knowledgebase that serves as input for subsequent steps. In particular, DRE connects to the underlying DBMS to extract schema information (most data sources support some form of Call-Level Interface such as JDBC). The schema information from the database is semantically enhanced using clues extracted by the semantic analyzer from available application code, business reports, and, in the future, perhaps other electronically available information that may encode business data such as e-mail correspondence, corporate memos, etc. It has been our experience (through visits with representatives from the construction and

[¥] Rinker School of Building Construction, University of Florida, Gainesville, FL 32634-6134 manufacturing domains) that such application code exists and can be made available electronically. Second, the semantically enhanced legacy source schema must be mapped into the domain model (DM) used by the application(s) that want(s) to access the legacy source. This is done using a schema mapping process that produces the mapping rules between the legacy source schema and the application domain model. In addition to the domain model, the schema mapper also needs access to the domain ontology (DO) describing the model.

Finally, the extracted legacy schema and the mapping rules provide the input to the wrapper generator (not shown), which produces the source wrapper. In this paper, we focus on our implementation of the DRE algorithm.

3 Data Reverse Engineering

Data reverse engineering (DRE) is defined as the application of analytical techniques to one or more legacy data sources to elicit structural information (e.g., term definitions, schema definitions) from the legacy source(s) in order to improve the database design or produce missing schema documentation. So far in SEEK, we are applying DRE to relational databases only. However, since the relational model has only limited semantic expressability, in addition to the schema, our DRE algorithm generates an E/R-like representation of the entities and relationships that are not explicitly defined in the legacy schema (but which exist implicitly). Our approach to data reverse engineering for relational sources is based on existing algorithms by Chiang [1, 2] and Petit [8]. However, we have improved their methodologies in several ways, most importantly to reduce the dependency on human input and to eliminate some of the limitations of their algorithms (e.g., consistent naming of key attributes, legacy schema in 3-NF).

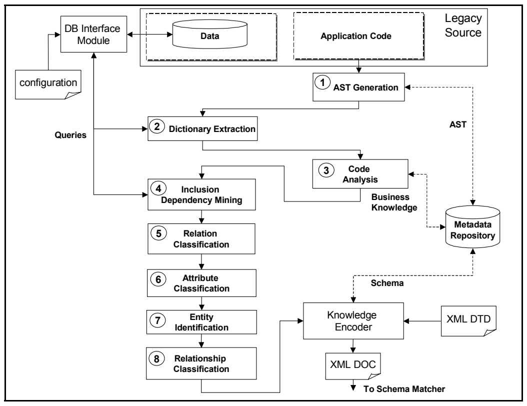

Figure 2: Conceptual overview of the DRE algorithm.

Our DRE algorithm is divided into schema extraction and semantic analysis, which operate in interleaved fashion. An overview of the two algorithms, which are comprised of eight steps, is shown in Figure 2. In addition to the modules that execute each of the eight steps, the architecture in Figure 3 includes three support components: the configurable *Database Interface Module* (upper-right hand corner), which provides connectivity to the underlying legacy source. Note that this component is the ONLY source-specific component in the architecture: in order to perform knowledge extraction from different sources, only the interface module needs to be changed. The *Knowledge Encoder* (lower right-hand corner) represents the extracted knowledge in the form of an XML document so that it can be shared with other components in the SEEK architecture (e.g., the semantic matcher). The *Metadata Repository* is internal to DRE and used to store intermediate run-time information needed by the algorithms including user input parameters, the abstract syntax tree for the code (e.g., from a previous invocation), etc.

We now highlight each of the eight steps and related activities outlined in Figure 3 using an example from our construction supply chain testbed. For a detailed description of our algorithm, refer to [3]. For simplicity, we assume without lack of generality or specificity that only the following relations exist in the MS-Project application, which will be discovered using DRE (for a description of the entire schema refer to [5]):

```
MSP-Project [PROJ_ID, ...]
MSP-Availability[PROJ_ID, AVAIL_UID, ...]
MSP-Resources [PROJ_ID, RES_UID,  ...]
MSP-Tasks [PROJ_ID, TASK_UID,  ...]
MSP-Assignment [PROJ_ID, ASSN_UID,  ...]
```

In order to illustrate the code analysis and how it enhances the schema extraction, we refer the reader to the following C code fragment representing a simple, hypothetical interaction with the MS Project database.

```
char *aValue, *cValue;
int flag = 0;
int bValue = 0;
EXEC SQL SELECT A,C INTO :aValue, :cValue
FROM Z WHERE B = :bValue;
if (cValue < aValue)
      {   flag = 1; }
printf("Task Start Date %s ", aValue);
printf("Task Finish Date %s ", cValue);
```

Step 1: AST Generation

We start by creating an Abstract Syntax Tree (AST) shown in Figure 3. The AST will be used by the semantic analyzer for code exploration during step 3. Our objective in AST generation is to be able to associate "meaning" with program variables. Format strings in input/output statements contain semantic information that can be associated with the variables in the input/output statement. This program variable in turn may be associated with a column of a table in the underlying legacy database.

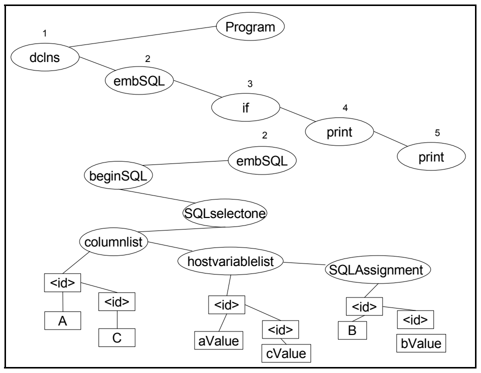

Figure 3: Application-specific code analysis via AST decomposition and code slicing. The direction of slicing is backwards (forward) if the variable in question is in an output (resp. input or declaration) statement.

Step 2. Dictionary Extraction.

The goal of step 2 is to obtain the *relation* and *attribute names* from the legacy source. This is done by querying the data dictionary, stored in the underlying database in the form of one or more system tables. Otherwise, if primary key information cannot be retrieved directly from the data dictionary, the algorithm passes the set of candidate keys along with predefined "rule-out" patterns to the code analyzer. The code analyzer searches for these patterns in the application code and eliminates those attributes from the candidate set, which occur in the rule-out pattern. The rule-out patterns, which are expressed as SQL queries, occur in the application code whenever programmer expects to select a SET of tuples. If, after the code analysis, not all primary key can be identified, the reduced set of candidate keys is presented to the user for final primary key selection.

Result. In the example DRE application, the following relations and their attributes were obtained from the MS-Project database:

```
MSP-Project [PROJ_ID, ...]
MSP-Availability[PROJ_ID, AVAIL_UID, ...]
MSP-Resources [PROJ_ID, RES_UID,  ...]
MSP-Tasks [PROJ_ID, TASK_UID,  ...]
MSP-Assignment [PROJ_ID, ASSN_UID,  ...]
```

Step 3: Code Analysis

The objective of step 3, code analysis, is twofold: (1) augment entities extracted in step 2 with domain semantics, and (2) identify business rules and constraints not explicitly stored in the database, but which may be important to the wrapper developer or application program accessing the legacy source. Our approach to code analysis is based on code analysis, which includes slicing [4] and pattern matching [7].

The first step is the *pre-slicing*. From the AST of the application code, the pre-slicer identifies all the nodes corresponding to input, output and embedded SQL statements. It appends the statement node name, and identifier list to an array as the AST is traversed in pre-order. For example, for the AST in Figure 3, the array contains the following information depicted in Table 1. The identifiers that occur in this data structure maintained by the pre-slicer form the set of slicing variables.

Table 1: Information maintained by the pre-slicer.

Node number	Statement	Text String (for print nodes)	Identifiers	Direction of Slicing
2	embSQL (Embedded SQL node)	-----	aValue cValue	Backwards

The code slicer and analyzer, which represent steps two and three respectively, are executed once for each slicing variable identified by the pre-slicer. In the above example, the slicing variables that occur in SQL and output statements are aValue and cValue. The direction of slicing is fixed as backwards or forwards depending on whether the variable in question is part of a output (backwards) or input (forwards) statement. The slicing criterion is the exact statement (SQL or input or output) node that corresponds to the slicing variable.

During code slicing sub-step we traverse the AST for the source code and retain only those nodes that have an occurrence of the slicing variable in sub-tree. This results in a reduced AST, which is shown in Fig. 4.

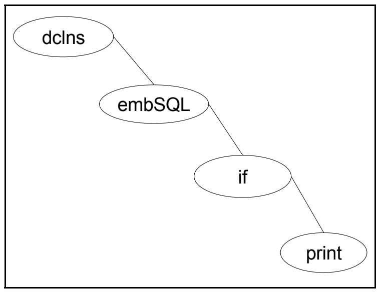

Figure 4: Reduced AST.

During the analysis sub-step, our algorithm extracts the information shown in Table 2, while traversing the reduced AST in pre-order.

1. If a *dcln* node is encountered, the data type of the identifier can be learned.
2. *embSQL* contain the mapping information of identifier name to corresponding column name and table name in the database.
3. *Printf/scanf* nodes contain the mapping information from the text string to the identifier. In other words we can extract the 'meaning' of the identifier from the text string.

Table 2: Information inferred during the analysis sub-step.

Identifier Name	Meaning	Possible Business Rule

aValue	Task Start Date	if (cValue < aValue)
		{
		}

cValue	Task Finish Date	if (cValue < aValue)
		{
		}

Data type	**Column Name in Source**	**Table Name in Source**
Char * => string	A	Z
Char * => string	C	Z

The results of analysis sub-step are appended to a result report file. After the code slicer and analyzer have been invoked on every slicing variable identified by the pre-slicer, the results report file is presented to the user. The user can base his decision of whether to perform further analysis based on the information extracted so far. If the user decides not to perform further analysis, code analysis passes control to the inclusion dependency detection module.

It is important to note, that we identify enterprise knowledge by matching templates against code fragments in the AST. So far, we have developed patterns for discovering business rules which are encoded in loop structures and/or conditional statements and mathematical formulae, which are encoded in loop structures and/or assignment statements. Note, the occurrence of an assignment statement itself does not necessarily indicate the presence of a mathematical formula, but the likelihood increases significantly if the statement contains one of the "slicing variables."

Step 4. Discovering Inclusion Dependencies.

After extraction of the relational schema in step 2, the goal of step 4 is to identify constraints to help classify the extracted relations, which represent both the real-world entities and the relationships among them. This is done using inclusion dependencies (INDs), which indicate the existence of inter-relational constraints including class/subclass relationships.

Let A and B be two relations, and X and Y be attributes or a set of attributes of A and B respectively. An inclusion dependency A.X << B.Y denotes that a set of values appearing in A.X is a subset of B.Y. Inclusion dependencies are discovered by examining all possible subset relationships between any two relations A and B in the legacy source.

Without additional input from the domain expert, inclusion dependencies can be identified in an exhaustive manner as follows: for each pair of relations A and B in the legacy source schema, compare the values for each non-key attribute combination X in B with the values of each candidate key attribute combination Y in A (note that X and Y may be single attributes). An inclusion dependency B.X<<A.Y may be present if:

1. X and Y have same number of attributes.
2. X and Y must have pair wise domain compatibility.
3. $B.X \subseteq A.Y$

In order to check the subset criteria (3), we have designed the following generalized SQL query templates, which are instantiated for each pair of relations and attribute combinations and run against the legacy source:

```
C1 =                    C2 =
SELECT count (*)        SELECT count (*)
FROM R1                 FROM R2
WHERE U NOT IN          WHERE V NOT IN
    (SELECT V                (SELECT U
     FROM R2);               FROM R1);
```

If C1 is zero, we can deduce that there may exist an inclusion dependency R1.U << R2.V; likewise, if C2 is zero there may exist an inclusion dependency R2.V << R1.U. Note that it is possible for both C1 and C2 to be zero. In that case, we can conclude that the two sets of attributes U and V are equal.

The worst-case complexity of this exhaustive search, given N tables and M attributes per table (NM total attributes), is O(N2M2). However, we reduce the search space in those cases where we can identify equi-join queries in the application code (during semantic analysis). Each equi-join query allows us to deduce the existence of one or more inclusion dependencies in the underlying schema. In addition, using the results of the corresponding count queries we can also determine the "direction" of the dependencies. This allows us to limit our exhaustive searching to only those relations not mentioned in the extracted queries.

Result: Inclusion dependencies are as follows:

1 MSP_Assignment[Task_uid,Proj_ID] << MSP_Tasks [Task_uid,Proj_ID]
2 MSP_Assignment[Res_uid,Proj_ID] << MSP_Resources[Res_uid,Proj_ID]
3 MSP_Availability [Res_uid,Proj_ID] << MSP_Resources [Res_uid,Proj_ID]
4 MSP_Resources [Proj_ID] << MSP_Project [Proj_ID]
5 MSP_Tasks [Proj_ID] << MSP_Project [Proj_ID]
6 MSP_Assignment [Proj_ID] << MSP_Project [Proj_ID]
7 MSP_Availability [Proj_ID] << MSP_Project [Proj_ID]

The last two inclusion dependencies are removed since they are implicitly contained in the inclusion dependencies listed in lines 2, 3 and 4 using the transitivity relationship.

Step 5. Classification of the Relations.

When reverse-engineering a relational schema, it is important to understand that due to the limited expressability of the relational model, all real-world entities are represented as relations irrespective of their types and role in the model. The goal of this step is to identify the different "types" of relations, some of which correspond to actual real-world entities while others represent relationships among them.

In this step all the relations in the database are classified into one of four types – strong, regular, weak or specific. Identifying different relations is done using the primary key information obtained in step 2 and the inclusion dependencies from step 4. Intuitively, a strong entity-relation represents a real-world entity whose members can be identified exclusively through its own properties. A weak entity-relation represents an entity that has no properties of its own that can be used to identify its members. In the relation model, the primary keys of weak entity-relations usually contain primary key attributes from other (strong) entity-relations. Both regular and specific relations are relations that represent relationships between two entities in the real world (rather then the entities themselves). However, there are instances when not all of the entities participating in an (n-ary) relationship are present in the database schema (e.g., one or more of the relations were deleted as part of the normal database schema evolution process). While reverse engineering the database, we identify such relationships as special relations.

Result:

Strong Entities: `MSP_Projects`
Weak Entities: `MSP_Resources, MSP_Tasks, MSP_Availability`
Regular Relationship: `MSP-Assignment`

Step 6. Classification of the Attributes.

We classify attributes as (a) PK or FK (from DRE-1 or DRE-2), (b) Dangling or General, or (c) Non-Key (rest).

Result: Table 3 illustrates attributes obtained from the example legacy source.

Table 3. Example of attribute classification from MS-Project legacy source.

	PKA	DKA	GKA	FKA	NKA
MS-Project	Proj_ID				All Remaining Attributes
MS-Resources	Proj_ID	Res_uid			
MS-Tasks	Proj_ID	Task_uid			
MS-Availability	Proj_ID	Avail_uid		Res_uid+ Proj_ID	
MS-Assignment	Proj_ID		Assn_uid	Res_uid+ Proj_ID, Task_uid + Proj_ID	

Step 7. Identify Entity Types.

Strong (weak) entity relations obtained from step 5 are directly converted into strong (resp. weak) entities.

Result: The following entities were classified:

Strong entities:

MSP_Project with Proj_ID as its key.

Weak entities:

MSP_Tasks with Task_uid as key and MSP_Project as its owner.

MSP_Resources with Res_uid as key and MSP_Project as its owner.

MSP_Availability with Avail_uid as key and MSP_Resources as owner.

Step 8. Identify Relationship Types.

The inclusion dependencies discovered in step 4 form the basis for determining the relationship types among the entities identified above. This is a two-step process:

1. Identify relationships present as relations in the relational database. The relation types (regular and specific) obtained from the classification of relations (Step 5) are converted into relationships. The participating entity types are derived from the inclusion dependencies. For completeness of the extracted schema, we may decide to create a new entity when conceptualizing a specific relation.

   The cardinality between the entities is M:N.
2. Identify relationships among the entity types (strong and weak) that were not present as relations in the relational database, via the following classification.
   - *IS-A relationships* can be identified using the PKAs of strong entity relations and the inclusion dependencies among PKAs. The cardinality of the IS-A relationship between the corresponding strong entities is 1:1.
   - *Dependent relationship*: For each weak entity type, the owner is determined by examining the inclusion dependencies involving the corresponding weak entity-relation. The cardinality of the dependent relationship between the owner and the weak entity is 1:N.

     Aggregate relationships: If the foreign key in any of the regular and specific relations refers to the PKA of one of the strong entity relations, an aggregate relationship is identified. The cardinality is either 1:1 or 1:N.
   - *Other binary relationships*: Other binary relationships are identified from the FKAs not used in identifying the above relationships. If the foreign key contains unique values, the cardinality is 1:1, else the cardinality is 1:N.

Result:

We discovered 1:N binary relationships between the following weak entity types:

Between MSP_Project and MSP_Tasks
Between MSP_Project and MSP_Resources
Between MSP_Resources and MSP_Availabilty

Since two inclusion dependencies involving MSP_Assignment exist (i.e., between Task and Assignment and between Resource and Assignment), there is no need to define a new entity. Thus, MSP_Assignment becomes an M:N relationship between MSP_Tasks and MSP_Resources.

At the end of Step 8, DRE has extracted the following schema information from the legacy database:

- Names and classification of all entities and attributes.
- Primary and foreign keys.
- Data types.
- Simple constraints (e.g., unique) and explicit assertions.
- Relationships and their cardinalities.
- Business rules

A conceptual overview of the extracted schema is represented by the entity-relationship diagram shown in Figure 5 (business rules not shown), which is an accurate representation of the information in encoded in the original MS Project schema.

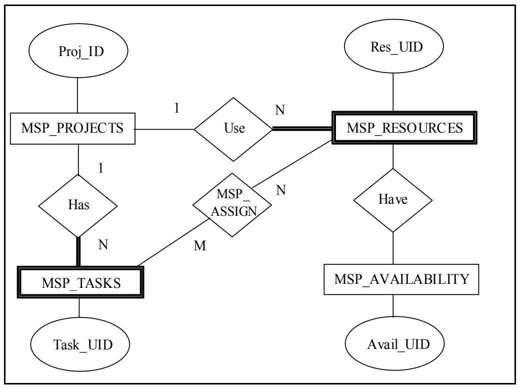

Figure 5: E/R diagram representing the extracted schema.

4 STATUS AND FUTURE WORK

We have manually tested our approach for a number of scenarios and domains (including construction, manufacturing and health care) to validate our knowledge extraction algorithm and to estimate how much user input is required. In addition, we have also conducted experiments using nine different database applications that were created by students during course projects. The experimental results so far are encouraging: the DRE algorithm was able to reverse engineer all of the sample legacy sources encountered so far. When coupled with semantic analysis, human input is reduced compared to existing methods. Instead the user is presented with clues and guidelines that lead to the augmentation of the schema with additional semantic knowledge.

The SEEK prototype is being extended using sample data from a large building construction project on the University of Florida campus in cooperation with the manager, Centex Rooney Inc., and several subcontractors or suppliers. This data testbed will support much more rigorous testing of the SEEK toolkit. Other plans for the SEEK toolkit are:

- Develop a formal representation for the extracted knowledge.
- Develop a matching tool capable of producing mappings between two semantically related yet structurally different schemas. Currently, schema matching is performed manually, which is a tedious, error-prone, and expensive process.
- Integrate SEEK with a wrapper development toolkit to determine if the extracted knowledge is sufficiently rich semantically to support compilation of legacy source wrappers for our construction testbed.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under grant numbers CMS-0075407 and CMS-0122193. The authors also thank Dr. Raymond Issa for his valuable comments and feedback on a draft of this paper.

REFERENCES

[1] R. H. Chiang, "A knowledge-based system for performing reverse engineering of relational database," *Decision Support Systems*, **13**, pp. 295-312, 1995.

[2] R. H. L. Chiang, T. M. Barron, and V. C. Storey, "Reverse engineering of relational databases: Extraction of an EER model from a relational database," *Data and Knowledge Engineering*, **12**:1, pp. 107-142., 1994.

[3] J. Hammer, M. Schmalz, W. O'Brien, S. Shekar, and N. Haldavnekar, "Knowledge Extraction in the SEEK Project," University of Florida, Gainesville, FL 32611-6120, Technical Report TR-0214, June 2002.

[4] S. Horwitz and T. Reps, "The use of program dependence graphs in software engineering," in *Proceedings of the Fourteenth International Conference on Software Engineering*, Melbourne, Australia, 1992.

[5] Microsoft Corp., "Microsoft Project 2000 Database Design Diagram", http://www.microsoft.com/office/project/prk/2000/Download/VisioHTM/P9_dbd_frame.htm.

[6] W. O'Brien, R. R. Issa, J. Hammer, M. S. Schmalz, J. Geunes, and S. X. Bai, "SEEK: Accomplishing Enterprise Information Integration Across Heterogeneous Sources," *ITCON - Journal of Information Technology in Construction*, 2002.

[7] S. Paul and A. Prakash, "A Framework for Source Code Search Using Program Patterns," *Software Engineering*, **20**:6, pp. 463-475, 1994.

[8] J.-M. Petit, F. Toumani, J.-F. Boulicaut, and J. Kouloumdjian, "Towards the Reverse Engineering of Denormalized Relational Databases," in *Proceedings of the Twelfth International Conference on Data Engineering (ICDE)*, New Orleans, LA, pp. 218-227, 1996.