SQL2XMI: Reverse Engineering of UML-ER Diagrams from Relational Database Schemas

Manar H. Alalfi James R. Cordy Thomas R. Dean
School of Computing, Queen's University, Kingston, Canada
{alalfi, cordy, dean}@cs.queensu.ca

Abstract

Data modeling is an essential part of the software development process, and together with application modeling forms the core of the model-driven approach to software engineering. While UML is considered the standard for application modeling, there is really no corresponding open standard for data modeling. In this paper, we propose an approach and a tool to help bridge the gap between application and data modeling based on source transformation technology. The tool, called SQL2XMI, automatically transforms an SQL schema into a UML-ER model expressed in XML Meta Interchange (XMI) 2.1. By bringing the data model to the UML world, both data and application models can be manipulated using the same UML-based tools.

1 Introduction

Model-driven software development generally begins with either application modeling or data modeling. In the application modeling domain, several Object Oriented modeling notations were combined in the early 1990s to produce the Unified Modeling Language (UML [15]). It has become the open standard notation for the specification and design of large software systems. Entity-Relationship diagrams (ER [6]) are usually used for data modeling, and in particular for relational databases. However, ER modeling does not define a standard graphical syntax, and there is no open standard for representing data objects. The original notation used by Chen [6] is widely used in academic texts and journals, but is rarely seen in either CASE tools or non-academic publications. Currently, the more common notations are Bachman [3], crow's foot [19], and IDEFIX [10]. This diversity in the underlying notations leads to data modeling that is tightly coupled to the tool of a specific vendor. From a model-driven engineering (MDE) perspective there is no general way to define transformations that either generate or consume such data models.

UML's generality can assist in the unification of all areas of expertise into a unified platform. This generality is in part due to *profiles*, a standardized set of extensions and constraints that tailors UML to particular uses [16]. Unfortunately, thus far there is no standard UML data modeling profile. Some UML vendors and users have defined their own UML profiles, but each has made their own interpretation and trade-offs, and all are UML 1.x based.

In this paper we present an automated transformation from an SQL (DDL) schema to an open XMI 2.1 UML-adapted class model. The adapted model is a tailored UML class model to represent the basic ER diagram components, including entities, attributes, relations, and primary keys. Our transformation technique is a novel one in that it is open, non-vendor specific, and targeted at the standard UML 2.1 exchange format, XMI 2.1. Although comparable commercial transformations exist, they are closed technologies targeted at formats tightly coupled to the vender's tools, hindering portability and preventing users from choosing their preferred tools in the development process.

While so far a prototype, our tool can recover a UML-based ER diagram from a SQL DDL schema and visualize it using any UML tool that supports the import of XMI 2.1 exchange format. The tool can be easily adapted to be compatible with different implementations of the SQL DDL notation, as well as different XMI 2.x versions. Using our method an open reverse transformation from XMI 2.x to an SQL DDL schema can also be easily implemented.

2 Previous Work

Ambler [2], Gorp [12], Gronik[11], and Silingas and Kaukenas [18] propose UML data modeling profiles to integrate with application modeling. Their profiles are a good partial solution, but they support only UML 1.x, do not address interoperability, and do not support the need for open transformations to generate and consume the models.

Many commercial tools, such as Rational Data Architect(RDA), Rational Software Architect(RSA) [5] and MagicDraw [13] provide a transformation facility to import SQL DDL schemas to ER diagrams. However, the transformations used in these systems involve specific proprietary file formats, making portability difficult. In contrast, our tool supports direct transformation from SQL DDL to open XMI 2.1. Because our tool is open rather than proprietary, it can easily be adapted to support other XMI 2.x versions.

1095-1350/08 $25.00 © 2008 IEEE
DOI 10.1109/WCRE.2008.30

Relational Database Element	Entity Relation Diagram Element	UML Class Diagram XMI elements (tailored to ER diagram)						
		XMI Tag	XMI Type	XMI ID	XMI Name	XMI Extension or additional attributes		
Table	<<Entity>>	packagedElement	uml:Class	Table name	Table name	eAnnotations extension with a detail element	type	ecore: EStringToStringMapEntry
							id	Unique ID
							Key	"Entity"
Column	Attribute	ownedAttribute	uml:Property	Attribute Name	Attribute Name	Additional attributes	visibility	private|public|..
							type	Attribute Data Type
Primary Key	Attribute With extension (stereotype annotation) <<PK>>	ownedAttribute	uml:Property	Attribute Name	Attribute Name	Additional attributes	visibility	private|public|..
							type	Attribute Data Type
						eAnnotations extension with a detail element	type	ecore: EStringToStringMapEntry
							id	Unique ID
							Key	"PK"
Foreign Key	Attribute	ownedAttribute	uml:Property	Source Table Name_ Target Table Name	Target Table Name(PK)	Additional attributes	visibility	private|public|..
							type	Target Table Name
							association	AssociationID
Association	Relation	packagedElement	uml:Association	AssociationID	NULL	Additional attributes	memberEnd	String type containing the id of entities member in this relation

Table 1. Mapping MySql schema elements, ERD elements, and XMI 2.1 elements

Abu-Hamdeh et al. [1] reverse engineered SQL schemas to Prolog-style textual factbases. While their transformation recovers more information than our tool, it assumes the explicit definition of foreign key constraints. By contrast, our implementation works with more general schemas that may not explicitly define such constraints, and our target representation is the XMI 2.1 model interchange standard. The additional information recovered by Abu-Hamdeh et al. can be added to our tool, but is not required in our current program comprehension project.

Another related system is Chung and Hartford's XMI2SQL [7], which transforms an XMI file exported from a UML tool to an SQL implementation. An ER model based on the Ambler profile [2] is built in UML, and exported to XMI. XMI2SQL then transforms this data model to an SQL DDL schema. This tool complements our work on the forward engineering side.

There is a long history of reverse engineering of ER diagrams from databases, such as Premerlani and Blaha [17]. These approaches are more mature and handle more features by utilizing more input artifacts. Di Lucca et al. [9], Yang et al. [20] and Canfora et al. [4] all recover data models from the source code of data intensive applications. The purpose of our work is different, providing a lightweight translation of SQL DDL schemas to standard UML, a problem the OMG called for proposals for in 2005 [14].

3 SQL2XMI

In this first implementation, we have targeted MySQL, although extending to other SQL variants is straightforward. Our current prototype assumes that the DDL schema is in third normal form and that the primary key constraint is explicitly defined. The prototype reverse engineers all the basic elements of the ER diagram: the set of entities and their attributes, the primary key set, the foreign key set, and the relationships between them. In this initial version we do not infer entity types or cardinalities, although they can be inferred if required. The prototype was developed as part of a project in software comprehension of web applications for security analysis, and in this role, entity types and cardinalities are not a concern.

3.1 Implementation

Our implementation is based on source transformation technology, in which the target program, an XMI file, is viewed as a syntactic modification of the source program, the SQL DDL schema. For this purpose we use TXL [8], a programming language designed for source transformation. TXL has been used in many production applications with transformations involving billions of lines of source code.

The TXL transformation process normally consists of three parts: a context-free grammar for the source language to be manipulated, a set of context-free grammar overrides (extensions or changes) to the base grammar, and a set of rules to implement the transformation [8]. In our case, the base language is SQL DDL and the overrides add output forms to represent XMI. In particular, the input is parsed as a sequence of SQL statements.

3.2 Entities, Attributes and Relationships

The correspondence between SQL DDL schema elements and XMI 2.1 (i.e. UML 2.1) elements implemented by the transformation is shown in Table 1. The function *GenerateERDElements*, shown in Figure 1, is responsible for the bulk of the transformation, transforming each SQL table definition, including all of its attributes, relations, and primary keys, to the corresponding XMI 2.1 elements. It is called once for each SQL table definition (*TableStructure*). Because SQL tables do not have a simple one-to-one relationship with the corresponding elements of the UML

```
% Generate the UML ER XMI element for each table column
function GenerateERDElements
            AllTableStructures [repeat MySQLStatement]
            TableStructure [MySQLStatement]
  deconstruct TableStructure
    CREATE TABLE TN[id](ColList [list createDefinition]);
  construct Tname [stringlit]
    _ [quote TN]
  construct SetOfAllPK [repeat XMItoken]
    _ [findAllPKL each AllTableStructures]
  construct SetOfPK [list index_col_name]
    _ [findPKL each ColList]
  construct eAnnotationsID [stringlit]
    _ [quote TN ] [+ "EAnnotation" ]
  construct detailsID [stringlit]
    _ [quote TN ] [+ "_Entity" ]
  construct XMI_ERD_Entity [repeat XMItoken]
    <packagedElement
        xmi:type="uml:Class" xmi:id=Tname name=Tname>
      <xmi:Extension
          extender="http://www.eclipse.org/emf/2002/Ecore">
        <eAnnotations xmi:type="ecore:EAnnotation"
           xmi:id= eAnnotationsID
           source="http://www.eclipse.org/uml2/2.0.0/UML">
          <details xmi:type="ecore:EStringToStringMapEntry"
            xmi:id= detailsID key="Entity"/>
        </eAnnotations>
      </xmi:Extension>
  construct packagedElementCloseT [repeat XMItoken]
     </packagedElement>
  construct XMI_ERD [repeat XMItoken]
    XMI_ERD_Entity
     [createEntityAttrib Tname SetOfAllPK each ColList]
     [constructFKside_Relation TableStructure each SetOfAllPK]
     [constructRelations TN AllTableStructures each SetOfPK]
     [. packagedElementCloseT]
     [constructAss TN SetOfPK each AllTableStructures]
  replace * [repeat XMItoken]
    % tail of output
  by
    XMI_ERD
end function
```

Figure 1. GenerateERDElements function

model and the transformation of each table depends on information in other tables, the main rule also passes all of the SQL tables in a separate parameter (*AllTableStructures*)

The function begins by breaking the `create table` statement into its syntactic parts so they can be accessed and manipulated separately. It extracts the table name and columns. Our implementation involves 16 separate transformation rules. We discuss three of the critical transformations in detail, and briefly outline the overall transformation process. The other rules in our transformation are similar.

The TXL constructor *SetOfAllPK* collects a list of all *(primary key, table name)* pairs based on the explicit primary key constraint statements in the table definitions. This information is used later in the rule to infer relationships for the current table. The constructor *SetOfPK* collects another list of all the primary keys defined in the current table. The constructor XMI_ERD_Entity creates the XMI representation of the entity for the table, consisting of an XMI *packagedElement* element of type *uml:Class* that is annotated as an *Entity*. The constructor XMI_ERD then uses a number of subrules to flesh out this initial ER representation by adding attributes, foreign keys and relationships to yield the entire translation of the SQL table definition.

```
% Generate the UML ER diagram XMI element
% for each table column
function createEntityAttrib  STableName [stringlit]
      PKL [ repeat XMItoken] Colm [createDefinition]
   deconstruct Colm
      ColName [colm_name] ColDef [col_def]
   deconstruct * [dataType] ColDef
      DT [dataType
   % isPKAttrib checks if the column is primary key
   % and annotates it with PK stereotype if so
   construct ModefidColName [repeat XMItoken]
      _ [isPKAttrib STableName ColName each PKL]
   construct AttribName [stringlit]
      _ [quote ColName]
   construct AttribDT [stringlit]
      _ [quote DT]
   construct AttribDef [repeat XMItoken]
      <ownedAttribute xmi:type="uml:Property"
         xmi:id= AttribName name= AttribName
         visibility="private">
   construct Closingtag [repeat XMItoken]
      </ownedAttribute>
   replace * [repeat XMItoken]
      % tail of output
   by
      AttribDef [.ModefidColName] [.Closingtag]
end function
```

Figure 2. The createEntityAttribute function

3.3 Attributes

For each table column, the *GenerateERDElements* function uses the function *createEntityAttribute*, shown in Figure 2. This function generates an XMI *ownedAttribute* of type *uml:Property* to represent the table column. It uses a deconstruct to capture the column's name(*ColName*), definition (*ColDef*), and data type (*DT*). The following construct uses the function *IsPKAttrib* to determine if the column is a primary key of the table and to annotate it with the primary key stereotype if so. The rest of *GenerateERDElements* creates the *ownedAttribute* representation of the column in XMI, embeds the primary key stereotypes and appends the result to the XMI representation of the table.

The transformation subrule *IsPKAttrib* identifies table primary keys by checking if the table column *ColN* and its table *STableName* are present in the set of all primary keys that was collected in *GenerateERDElements*. If so, an XMI primary key stereotype *eAnnotation* element that corresponds to the matched table column is returned by the function. The annotation is added to the XMI representation of the column by *GenerateERDElements*.

3.4 Relationships and Associations

Following the transformation of table columns, the *GenerateERDElements* function uses similar transformation subrules to handle foreign keys and relations. The subrule *constructFKside_Relation* identifies foreign keys as relation target by checking for an occurrence of each non primary key column in the set of all primary keys *SetOfAllPK*. For each primary key of another table T_i that matches, an XMI *ownedAttribute* element of type *uml:Property* to refer to

```
CREATE TABLE phpbb_forums (
 forum_id smallint(5) UNSIGNED NOT NULL,
 cat_id mediumint(8) UNSIGNED NOT NULL,
 forum_name varchar(150),
 forum_desc text,
 forum_status tinyint(4) DEFAULT '0' NOT NULL,
 forum_order mediumint(8) UNSIGNED DEFAULT '1'
   NOT NULL,
 forum_posts mediumint(8) UNSIGNED DEFAULT '0'
   NOT NULL,
 forum_topics mediumint(8) UNSIGNED DEFAULT '0'
   NOT NULL,
 forum_last_post_id mediumint(8) UNSIGNED
   DEFAULT '0' NOT NULL,
 prune_next int(11),
 prune_enable tinyint(1) DEFAULT '0' NOT NULL,
 auth_view tinyint(2) DEFAULT '0' NOT NULL,
 auth_read tinyint(2) DEFAULT '0' NOT NULL,
 auth_post tinyint(2) DEFAULT '0' NOT NULL,
 auth_reply tinyint(2) DEFAULT '0' NOT NULL,
 auth_edit tinyint(2) DEFAULT '0' NOT NULL,
 auth_delete tinyint(2) DEFAULT '0' NOT NULL,
 auth_sticky tinyint(2) DEFAULT '0' NOT NULL,
 auth_announce tinyint(2) DEFAULT '0' NOT NULL,
 auth_vote tinyint(2) DEFAULT '0' NOT NULL,
 auth_pollcreate tinyint(2) DEFAULT '0'
    NOT NULL,
 auth_attachments tinyint(2) DEFAULT '0'
    NOT NULL,
 PRIMARY KEY (forum_id),
 KEY forums_order (forum_order),
 KEY cat_id (cat_id),
 KEY forum_last_post_id (forum_last_post_id));
```

```
CREATE TABLE phpbb_topics (
 topic_id mediumint(8) UNSIGNED NOT NULL auto_increment,
 forum_id smallint(8) UNSIGNED DEFAULT '0' NOT NULL,
 topic_title char(60) NOT NULL,
 topic_poster mediumint(8) DEFAULT '0' NOT NULL,
 topic_time int(11) DEFAULT '0' NOT NULL,
 topic_views mediumint(8) UNSIGNED DEFAULT '0' NOT NULL,
 topic_replies mediumint(8) UNSIGNED DEFAULT '0' NOT NULL,
 topic_status tinyint(3) DEFAULT '0' NOT NULL,
 topic_vote tinyint(1) DEFAULT '0' NOT NULL,
 topic_type tinyint(3) DEFAULT '0' NOT NULL,
 topic_first_post_id mediumint(8) UNSIGNED DEFAULT '0' NOT NULL,
 topic_last_post_id mediumint(8) UNSIGNED DEFAULT '0' NOT NULL,
 topic_moved_id mediumint(8) UNSIGNED DEFAULT '0' NOT NULL,
 PRIMARY KEY (topic_id),
 KEY forum_id (forum_id),
 KEY topic_moved_id (topic_moved_id),
 KEY topic_status (topic_status),
 KEY topic_type (topic_type));

CREATE TABLE phpbb_forum_prune (
 prune_id mediumint(8) UNSIGNED NOT NULL auto_increment,
 forum_id smallint(5) UNSIGNED NOT NULL,
 prune_days smallint(5) UNSIGNED NOT NULL,
 prune_freq smallint(5) UNSIGNED NOT NULL,
 PRIMARY KEY(prune_id),
 KEY forum_id (forum_id));

CREATE TABLE phpbb_categories (
 cat_id mediumint(8) UNSIGNED NOT NULL auto_increment,
 cat_title varchar(100),
 cat_order mediumint(8) UNSIGNED NOT NULL,
 PRIMARY KEY (cat_id),
 KEY cat_order (cat_order));
```

```
CREATE TABLE phpbb_auth_access (
 group_id mediumint(8) DEFAULT '0' NOT NULL,
 forum_id smallint(5) UNSIGNED DEFAULT '0'
    NOT NULL,
 auth_view tinyint(1) DEFAULT '0' NOT NULL,
 auth_read tinyint(1) DEFAULT '0' NOT NULL,
 auth_post tinyint(1) DEFAULT '0' NOT NULL,
 auth_reply tinyint(1) DEFAULT '0' NOT NULL,
 auth_edit tinyint(1) DEFAULT '0' NOT NULL,
 auth_delete tinyint(1) DEFAULT '0' NOT NULL,
 auth_sticky tinyint(1) DEFAULT '0' NOT NULL,
 auth_announce tinyint(1) DEFAULT '0' NOT NULL,
 auth_vote tinyint(1) DEFAULT '0' NOT NULL,
 auth_pollcreate tinyint(1) DEFAULT '0'
    NOT NULL,
 auth_attachments tinyint(1) DEFAULT '0'
    NOT NULL,
 auth_mod tinyint(1) DEFAULT '0' NOT NULL,
 PRIMARY KEY (group_id, forum_id),
 KEY group_id (group_id),
 KEY forum_id (forum_id));

CREATE TABLE phpbb_groups (
 group_id mediumint(8) NOT NULL auto_increment,
 group_type tinyint(4) DEFAULT '1' NOT NULL,
 group_name varchar(40) NOT NULL,
 group_description varchar(255) NOT NULL,
 group_moderator mediumint(8) DEFAULT '0'
   NOT NULL,
 group_single_user tinyint(1) DEFAULT '1'
   NOT NULL,
 PRIMARY KEY (group_id),
 KEY group_single_user (group_single_user));
```

Figure 3. Subset of the PhpBB 2.0 MySql schema

a relation of type T_i between the two tables is generated, where T_i is the name of the other table.

Next, *GenerateERDElements* uses the subrule *constructRelations* to identify foreign keys as relation source by checking for columns that are primary keys in this table and also occur as a column in another table. For each such foreign key, an XMI *ownedAttribute* element of type *uml:Property* is generated to refer to a relation of type T_i between the two tables.

Finally, *GenerateERDElements* uses the transformation subrule *constructAss* to create an XMI *packagedElement* of type *uml:Association* between the two tables involved in each relation generated by the previous two subrules.

The concatenation of the results of the subrules of *GenerateERDElements* forms the complete result transformation of the column to XMI, and the concatenation of the columns gives the result of the main transformation function, yielding the complete XMI 2.1 representation of the UML 2.1 ER diagram for the original SQL schema.

4 An Example: PhpBB

SQL2XMI is designed to serve an ongoing project in web application security analysis, in which reverse engineering is used to identify the application resources, permissions, and subjects that constitute the basic elements of a security model. Data models constitute one of the main sources of such information, and visualizing data models facilitates the process of understanding system structure.

In this context, we have evaluated SQL2XMI on the popular web bulletin board system PhpBB versions 2.0 and 3.0. SQL2XMI has been able to automatically recover ER diagrams for the data models of both versions. The results have helped us to understand the complex data model of this system and to recognize that the data model of PhpBB 3.0 has been completely restructured from the previous version. PhpBB 2.0 has 30 tables that generate a UML 2.1 ER diagram much too large to fit in this paper. A part of the schema is shown in Figure 3 We can see in Figure 4 that the entity $phpbb\_groups$ is involved in a relation with $phpbb\_auth\_access$ based on its primary key. Even in this simple example we can see that important information such as relations are not easily understood directly from the SQL schema. The process of comprehending the application at this level can be tedious work, especially for more complicated schemas. Visualizing the translated XMI file as an ER diagram as shown in Figure 4, however, presents the database schema in form that can be easily and quickly understood by all members of the development team.

5 Conclusions and Future Work

In this paper we have presented a source transformation technique to bridge the gap between data modeling and application modeling that can assist in the process of complex software comprehension and evolution. Our new open tool, SQL2XMI, automatically transforms an SQL DDL schema to a UML 2.1 ER diagram which can be visualized by any UML tool that supports XMI 2.1. Unlike other tools that reverse engineer to proprietary formats, SQL2XMI explicitly aims at open and flexible portability, requiring only the SQL DDL schema and targeting the official OMG XMI 2.1 UML representation. We have presented the details of our lightweight source transformation-based approach and an example of the application of our tool to recover an ER diagram for the popular internet bulletin board system PhpBB. The approach and mapping are unique to our work.

To bring our prototype tool to an industrial level, several

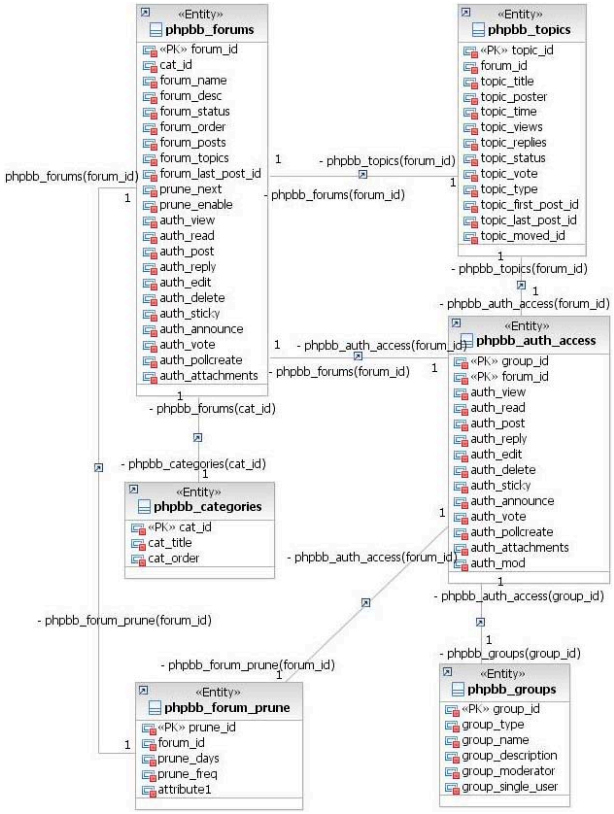

Figure 4. Example RSA visualization

improvements will be needed. It must be generalized to handle SQL database schemas other than MySQL and XMI 2.x versions other than 2.1. Using TXL gives us the ability to integrate handling of other implementations of the SQL standard quickly, simply by overriding the SQL grammar to add the forms of each vendor's specific extensions.

In this paper we have begun with just the MySql implementation of the SQL data definition language (DDL), leaving the improvement of the grammar file to include the data manipulation part (DML) and support for other vendors' implementations to future work. Our transformation also does not yet take advantage of all of the information available in the schema. Using a more comprehensive transformation rule set, we hope to recover a richer ER model.

Finally, while in this work we have concentrated on reverse engineering an existing MySql schema to a UML entity relationship diagram, in future we could use the same technique in the forward engineering direction, using the same technology to generate different SQL database implementations from an ER diagram designed using any UML toolset that supports XMI 2.1 export.

Acknowledgments

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

References

[1] R. Abu-Hamdeh, J. R. Cordy, and T. P. Martin. Schema translation using structural transformation. In *CASCON*, pages 202–215, 1994.

[2] S. Ambler. A UML profile for data modeling. www.agiledata.org: Techniques for Successful Evolutionary/Agile Database Development, 2006.

[3] C. W. Bachman. Data structure diagrams. *SIGMIS Database*, 1(2):4–10, 1969.

[4] G. Canfora, A. Cimitile, A. De Lucia, and G. A. Di Lucca. Decomposing legacy systems into objects: an eclectic approach. *Inf. & Soft. Tech.*, 43(6):401–412, 2001.

[5] D. T. Chang. Integrating Rational Software Architect with Rational Data Architect. IBM developerWorks, 2007.

[6] P. P.-S. Chen. The entity-relationship model—toward a unified view of data. *ACM Trans. Datab. Syst.*, 1(1):9–36, 1976.

[7] S. Chung and E. Hartford. Bridging the gap between data models and implementations: XMI2SQL. In *AICT/ICIW*, page 201, 2006.

[8] J. R. Cordy. The TXL source transformation language. *Sci. Comput. Program.*, 61(3):190–210, 2006.

[9] G. A. Di Lucca, A. R. Fasolino, and U. de Carlini. Recovering class diagrams from data-intensive legacy systems. In *ICSM*, pages 52–63, 2000.

[10] Federal Information Processing Standards. Publication 184, Integration Definition for Information Modeling (IDEFIX), *http://www.itl.nist.gov/fipspubs/idef1x.doc*.

[11] D. Gornik. UML data modeling profile. Technical report, IBM Rational Software Whitepaper TP 162 05/02, 2003.

[12] P. V. Gorp. UML profile for data modeling, *http://www.fots.ua.ac.be/~pvgorp/research/datamodelingprofile/*, 2007.

[13] No Magic, Inc. MagicDraw UML, *http://www.magicdraw.com*.

[14] Object Management Group (OMG). Request For Proposal Information Management Metamodel (IMM), *http://www.omg.org/docs/ab/05-12-02.pdf*. Technical report, 2005.

[15] Object Management Group (OMG). OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2, *http://www.omg.org/docs/formal/07-11-01.pdf*. Technical report, 2007.

[16] Object Management Group (OMG). UML Profile Catalog, *http://www.omg.org/technology/documents/profile_catalog.htm*. Technical report, 2008.

[17] W. J. Premerlani and M. R. Blaha. An approach for reverse engineering of relational databases. *Commun. ACM*, 37(5):42–49, 134, 1994.

[18] D. Silingas and S. Kaukenas. Applying UML for relational data modeling, *http://www.magicdraw.com/files/articles/Sep04%20Applying%20UML%20for%20Relational%20Data%20Modeling.htm*, 2004.

[19] T. J. Teorey, D. Yang, and J. P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Comput. Surv.*, 18(2):197–222, 1986.

[20] H. Yang and W. C. Chu. Acquisition of entity relationship models for maintenance-dealing with data intensive programs in a transformation system. *J. Inf. Sci. Eng.*, 15(2):173–198, 1999.