Extracting Semantic Metadata and Its Visualization

By *Dongwon Lee* and *Yousub Hwang*

Introduction

Reverse engineering examines the problem of understanding an existing system and recovering essential design specifications [5]. Chiang et al. define database reverse engineering (**DBRE**) as a process that obtains domain semantics about an existing database, then converts the schema from relational to conceptual, and finally represents the results as a conceptual schema [4]. The objectives of the DBRE process are to improve the understanding of data semantics [12], to mechanically reuse past development outcomes, to reduce maintenance cost and improve software flexibility [13], and to integrate several databases [3].

In this article, we propose a reverse engineering agent for the Conflict Resolution Environment Autonomous Mediation (CREAM) system, called the *Semantic Metadata Extracting & Visualizing Agent* (**SMEVA**) [15]. As a semi-autonomous agent that transforms relational schema to conceptual schema using the Unifying Semantic Model (USM) constructs [14], *SMEVA* has been developed to achieve four objectives:

- to provide semi-autonomous conceptual schema development,
- to avoid trivial errors in the development of conceptual schema,
- to assist users lacking prior knowledge of the relational schema,
- to improve efficiency in the semantic mediation process by reducing USM design time.

As interaction with users is crucial for capturing essential domain semantics [4], our system requires user involvement in transforming relations to the USM constructs.

We implemented the SMEVA agent using Java technology (JDK 1.3 including AWT and Swing) with a back-end Oracle 8i DBMS. In addition, we utilized the JDBC API to establish a connection with a relational database management system (RDBMS), which we use as our data source.

Unifying Semantic Model (USM)

The general task of database design is to map a given real world application into the formal data model of a given database management system (DBMS) [6]. The *Entity-Relationship* (**ER**) model, initially proposed by Chen in 1976 [2], has been widely accepted in the area of database conceptual modeling. Its fundamental modeling constructs are entities, relationships, and their associated attributes. However, the lack of initial modeling constructs has made it difficult to use the ER model as a conceptual schema representation [17].

Ram [14] developed an extended version of the traditional ER model, the *Unifying Semantic Model* (**USM**), which allows users to view real world situations in a more natural way. USM uses high-level abstractions -- such as classification, association, generalization, and aggregation -- found in traditional semantic models [7]. The USM constructs have explicit definitions that govern their relationships and semantic properties, including strong/weak/

interaction entity classes, super/subclasses, attributes, and domain classes, as well as interaction relationships. In addition, USM provides explicit constructs to represent constraints on generalization/specialization relationships. It also formally defines complex objects, such as groups/aggregates and composites, which can be used to capture a hierarchy of complex objects. Graphical representations of the constructs in USM are illustrated in Figure 1.

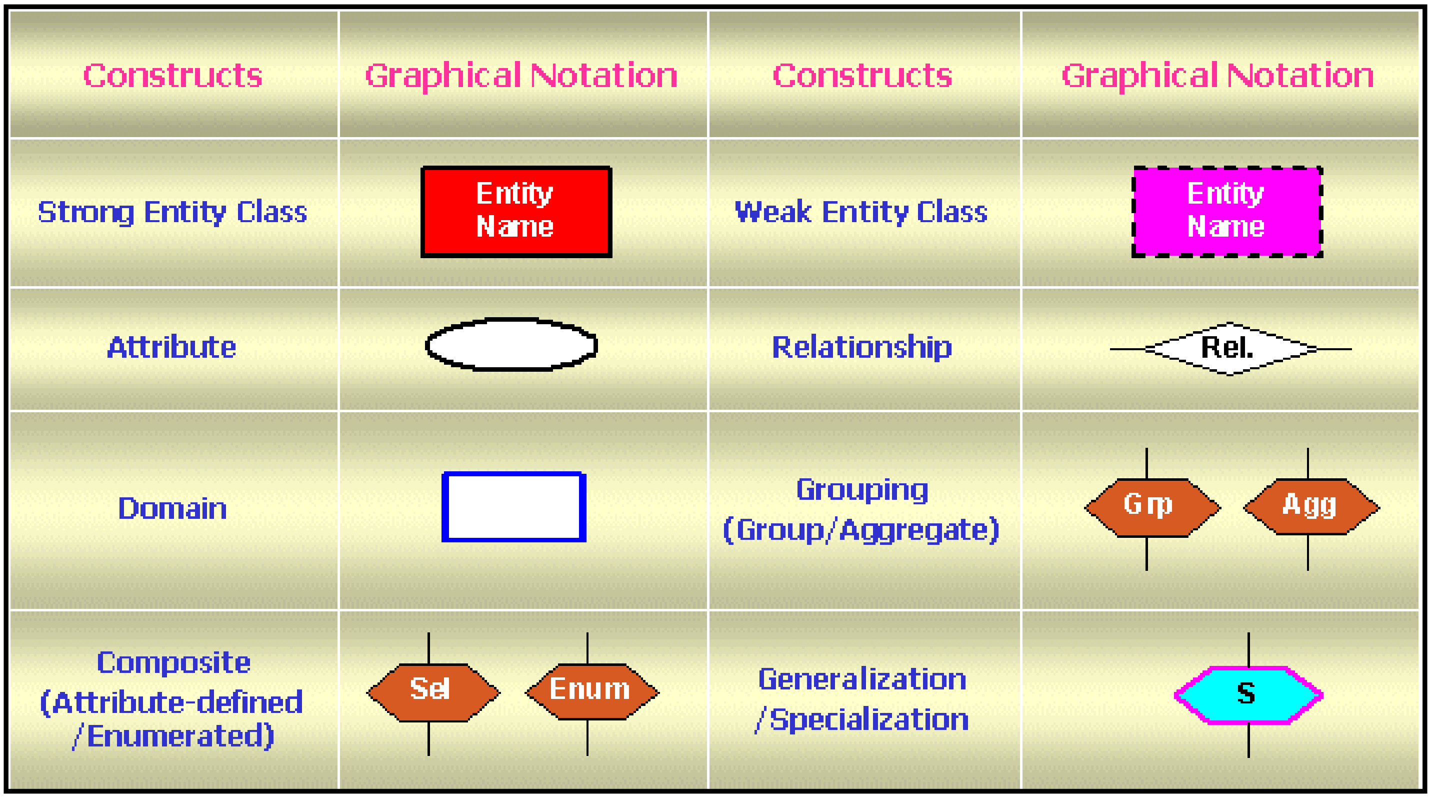


Figure 1: USM Constructs and Graphical Notations

CREAM

Conflict Resolution Environment Autonomous Mediation (**CREAM**)[15] is a software tool for information integration and semantic conflict detection and resolution that provides an interface to decision-making tools. It provides automatic managing of various semantic conflicts and thus facilitates semantic interoperability among homogeneous and heterogeneous geographic and non-geographic database systems. In order to provide a global query environment for distributed heterogeneous information sources, CREAM incorporates a three-layered architecture. The first layer, the **information source**, represents a number of distributed heterogeneous information sources (e.g., RDBMS). In the semantic mediation process, CREAM requires detailed metadata information, which is the description of a database, for each information source. Thus, a domain expert must be called upon to specify the metadata information for each information source by manually drawing a diagram of the conceptual schema using USM constructs in CREAM. The **local schema layer** is the second layer, which is a collection of the manually drawn conceptual schemas for the corresponding information sources. The final layer consists of a **federated schema and ontology**. The federated schema is a subset manually designed by the domain expert to unify all local schemas; the ontology is the context knowledge represented in a common vocabulary and used for automatically detecting and dynamically resolving various semantic conflicts among information sources. The detailed semantic metadata information is captured by the domain expert's illustration of the manual mapping process among federated schema, local schema, and the ontology. The architecture of the CREAM system is illustrated in Figure 2.

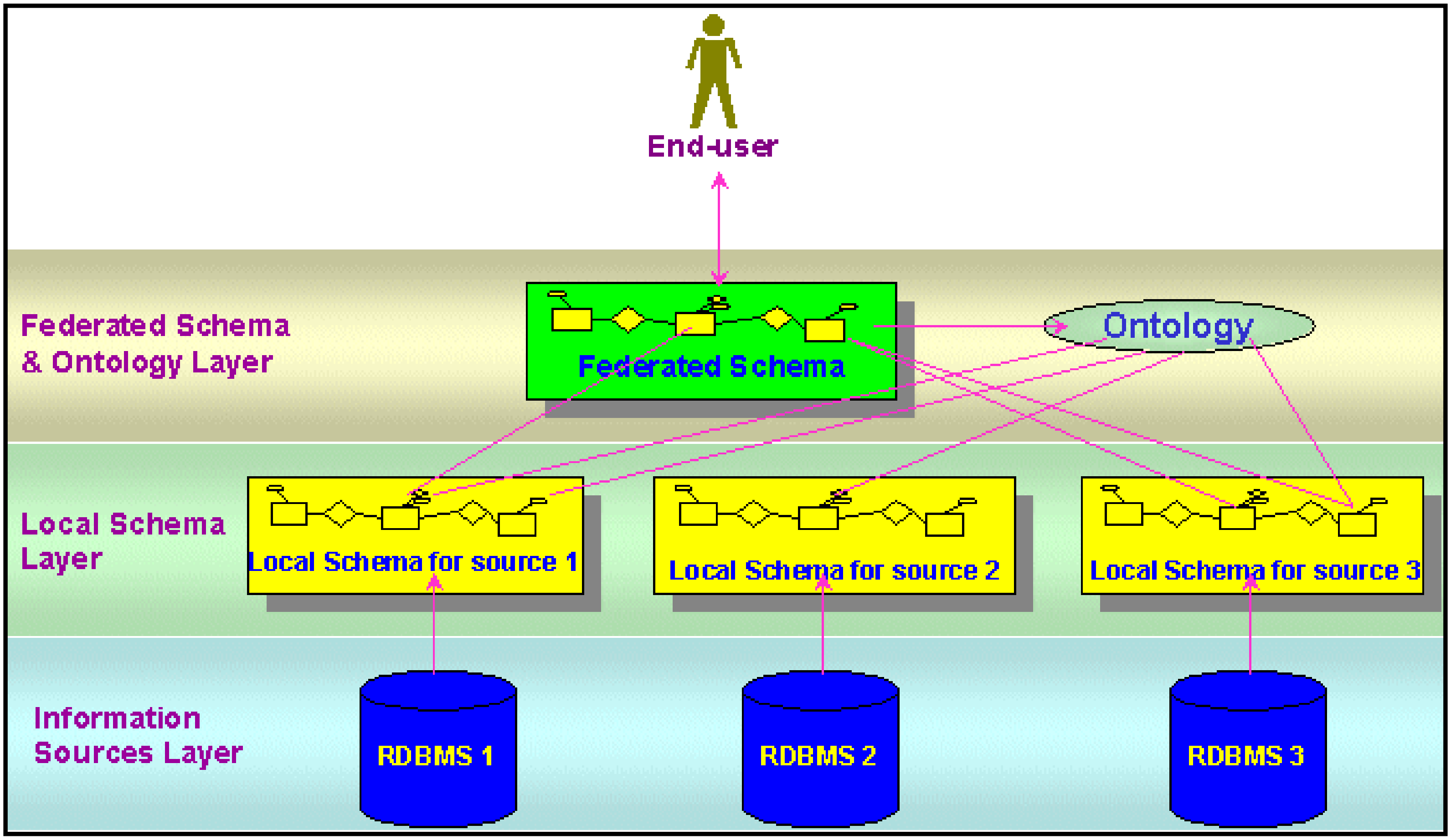


Figure 2: Three-Layered Semantic Mediation Architecture in the CREAM System

Database Reverse Engineering

Over the past few years, researchers have produced several papers detailing methods for transforming a relational database into a conceptual model [1, 3, 4, 6, 8, 9, 10, 12, 13, 16]. Each exhibits its own methodological characteristics, specific assumptions and inputs, and produces its own conceptual model.

Research	Assumption	Input	Output	Characteristics
Batini et al. (1992)	▪ BCNF or 3NF ▪ Attribute naming consistency ▪ No homonyms ▪ Specified PK & candidate key.	▪ Inclusion dependencies ▪ Relation schemes	▪ Entities ▪ Binary Relationships ▪ Categories ▪ Generalization	▪ Based on Navathe & Awong's paper but further simplified. ▪ Allowing a less normalized relation to be stored. ▪ Drawback of requiring semantic input earlier in the process.
Chiang et al. (1994, 1996)	▪ 3NF ▪ Attribute naming consistency ▪ No error on key attributes	▪ Relation schemes ▪ Data instances ▪ Inclusion dependencies	▪ Entities (Strong/Weak) ▪ Binary Relationship ▪ Generalization ▪ Aggregation	▪ Requiring knowledge about attribute name. ▪ Proposing a framework for the evaluation of DBRE methods. ▪ Clearly identifying the cases in which human input is required.
Davis & Arora (1987)	▪ 3NF ▪ No Homonyms & Synonyms	▪ Relation schemes ▪ Foreign key constraints	▪ Entity(-sets) ▪ Dangling Keys ▪ Binary Relationships	▪ Ignoring Inheritance. ▪ Aiming at an invertible transformation from relational schema to conceptual schema

(1987)	Synonyms	▪ Foreign key constraints	▪ Binary Relationships ▪ N-ary Relationships	from relational schema to conceptual schema.
Johanness on (1994)	▪ 3NF ▪ Domain independent queries	▪ Relation schemes ▪ Functional dependencies ▪ Inclusion dependencies	▪ Generalization ▪ Entities ▪ Binary Relationships	▪ Based on the well-established concepts of relational database theory. ▪ Drawback of needing all keys and inclusion dependencies. ▪ Simple and automatic mapping process.
Markowitz & Makowsky (1990)	▪ BCNF	▪ Relation schemes ▪ Key dependencies ▪ Referential Integrity dependencies	▪ Entity ▪ Binary Relationships ▪ Generalization ▪ Aggregation	▪ Presenting theoretically sound treatment of the mathematical basis. ▪ Not addressing the cascading optimization and poor database design. ▪ Requiring all key functional dependencies and key-based inclusion dependencies.
Navathe & Awong (1987)	▪ 3NF or some 2NF ▪ Attribute naming consistency ▪ No FK ambiguities ▪ Specified candidate keys	▪ Relation schemes	▪ Entities ▪ Binary Relationships ▪ Categories ▪ Cardinalities	▪ Drawback of requiring semantic input earlier in the process. ▪ Vulnerable to ambiguities in recognizing FK's. ▪ Resolving the most common situations rather than claim exhaustiveness.
Petit et al. (1996)	▪ 1NF ▪ Unique attributes	▪ Relation schemes ▪ Data instance & code	▪ Entities ▪ Relationships ▪ Generalization	▪ Coping with denormalized relational schemas in a DBRE process. ▪ Analyzing equi-join queries in application programs. ▪ No restriction on the naming of the attributes.
Premerlani & Blaha (1993, 1994)	▪ Non-3NF ▪ Semantic understanding of application	▪ Relation schemes ▪ Observed patterns of data	▪ Class ▪ Association ▪ Generalization ▪ Multiplicity ▪ Aggregation	▪ Requiring High level of human input. ▪ Providing guidelines for coping with design optimizations. ▪ Emphasizing analysis of candidate keys rather than primary keys.
Soutou (1997, 1998)	▪ No attribute naming uniqueness ▪ Unknown dependencies	▪ Data Schema ▪ Data Instance ▪ Data Dictionary	▪ Cardinality constraints on n-ary relationship	▪ Fully automating process for SQL 92 relational databases. ▪ Independent of the target semantic model.

Table 1: Summary of Reverse Engineering Research

Our research extends previous database reverse engineering research (refer to Table 1) as summarized here.

Our methodology utilizes information obtained from multiple sources including, analysis of relational schema, analysis of data instances, semantic understanding and heuristics, and reverse engineering algorithms. Therefore, the algorithms for our system, which describe how a conceptual schema is obtained from a relational database system, are obvious enough to be encoded into a reverse engineering agent. Furthermore, by analyzing data instances in the database, our system extracts the correct **cardinality constraints** (maximum and minimum number of relationship instance(s) in which an entity can participate) among the entity classes.

While the majority of previous research assumed the relations of the input database at least to be in 3NF, our research is based on the practical assumptions that there are no constraints on functional and inclusion dependencies or on attribute naming consistency or uniqueness. For modern relational databases where primary key and functional dependency constraints exist in the metadata (i.e., schema definition), these assumptions are not necessary.

Our system determines the kinds of domain semantics that must be obtained from an expert in the database domain. For instance, a domain expert is asked to provide the necessary semantics (e.g., attribute-defined, roster-defined, and set-defined subclass) when the reverse engineering agent detects a super/subclass relationship among entity classes.

Our system improves efficiency in the semantic mediation process by reducing USM design time, compared to the experimental results in Park's research [11]. In the following section, we describe our semantic metadata extraction rules and algorithms, and detail the implementation of the SMEVA agent.

Semantic Metadata Extraction and Visualization

In order to complete the database reverse engineering (DBRE) process satisfactorily, *SMEVA* is required to apply the schema conversion rules (refer to Table 2, Table 3, Table 4, and Table 5) in a logical sequence as illustrated in Figure 3.

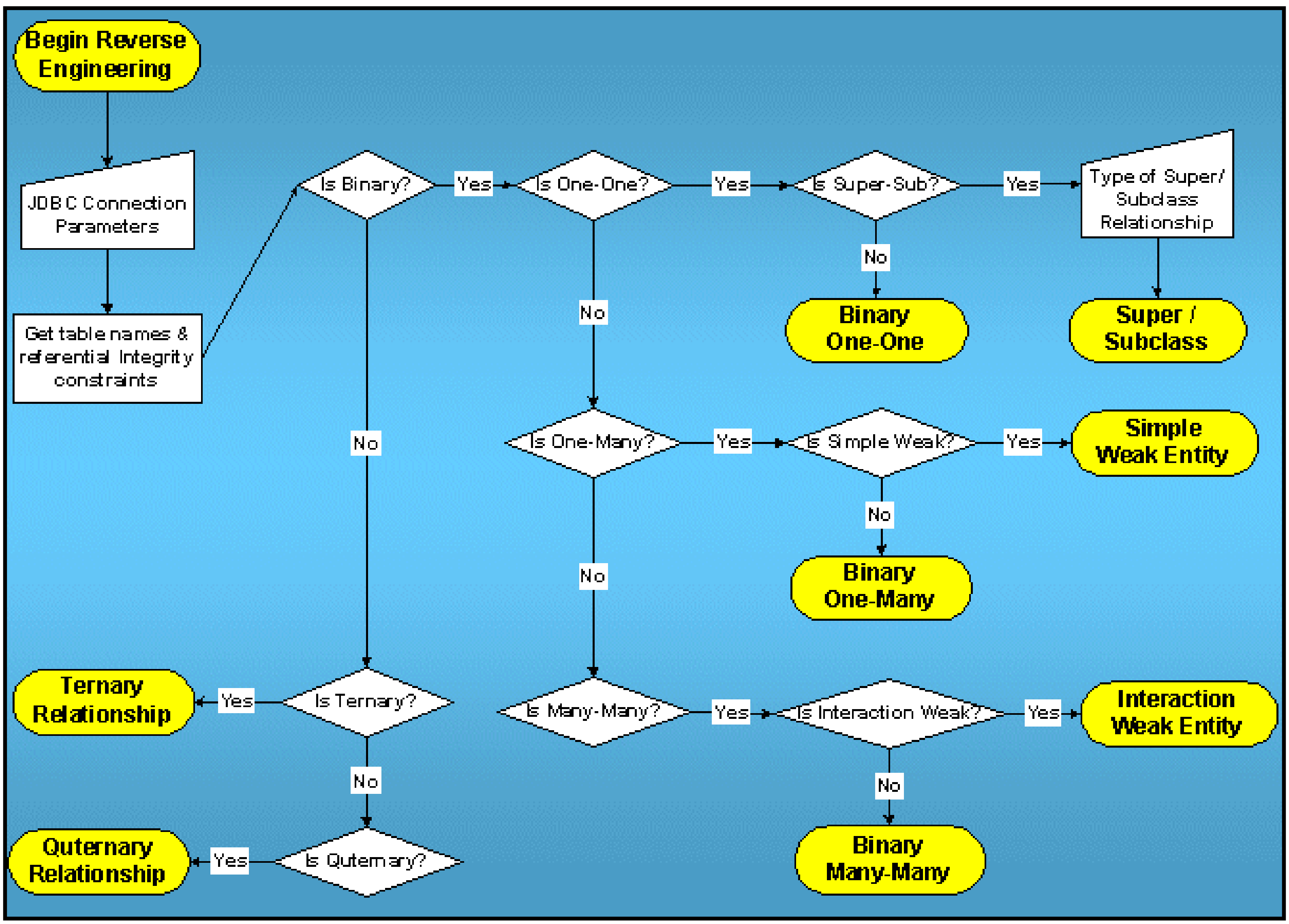


Figure 3: Logical Sequence of Reverse Engineering Process

SMEVA first collects JDBC connection parameters (i.e., host name, user name, password, and instance name) from the user, and makes a connection to the designated DBMS server. If a JDBC connection is completed, our system retrieves all relation names and referential integrity constraints using JDBC API methods, and then applies the reverse-engineering rules to determine the appropriate USM constructs. Our agent then checks the number of other relations referred to by a given name. If a given relation refers to one or two other relations, the system determines that a binary relationship exist. On the other hand, if a given relation refers to three or more other relations, then it is considered that a N-ary relationship exists (i.e., N participating entity classes in a relationship, such as ternary or quaternary) among the relations. Based on this initial decision, the system continues the reverse engineering process. In the following subsections, we explain each step of the algorithm in more detail.

Detection of Binary One-One Relationship & Super/Sub Class Relationship

One-One Relationship	▪Two relations refer to each other. ▪Foreign key in a relation refers to only one other relation and maximum cardinality constraint between two relations is One-One.
Super/Sub Class Relationship	The relationship between supertype and subtype is One (1,1) - One (0,1) cardinality and primary key in a relation (subtype) is entirely composed of foreign key of another relation (supertype).

Table 2: Reverse Engineering Rules: One-One & Super/Subclass

Let us assume that a RDBMS contains the relational schema illustrated in Figure 4. Retrieving all the names of relations and their referential integrity constraints from the designated RDBMS, *SMEVA* checks each referential integrity constraint one by one to complete the reverse engineering process. For instance, our agent discovers that the 'CREDIT_CARD' relation refers to the 'CUSTOMER' relation. A binary relationship is detected when a given a relation refers to one or two other relations.

CUSTOMER (<u>CustID</u>, Name, Address, Phone, Email, Type)

CREDIT_CARD (<u>CardNo</u>, CardType, ExpDate, NamePrinted, CustomerID)

EMPLOYEE (<u>EmpNo</u>, SSN, Name, Department)

SALARIED_PERSON (<u>EmpNo</u>, Salary, MaxSalary)

SALES_PERSON (<u>EmpNo</u>, CommisionRate, MaxCommRate)

<u>Referential Constraints:</u>

CREDIT_CARD.CustomerID → CUSTOMER.CustID

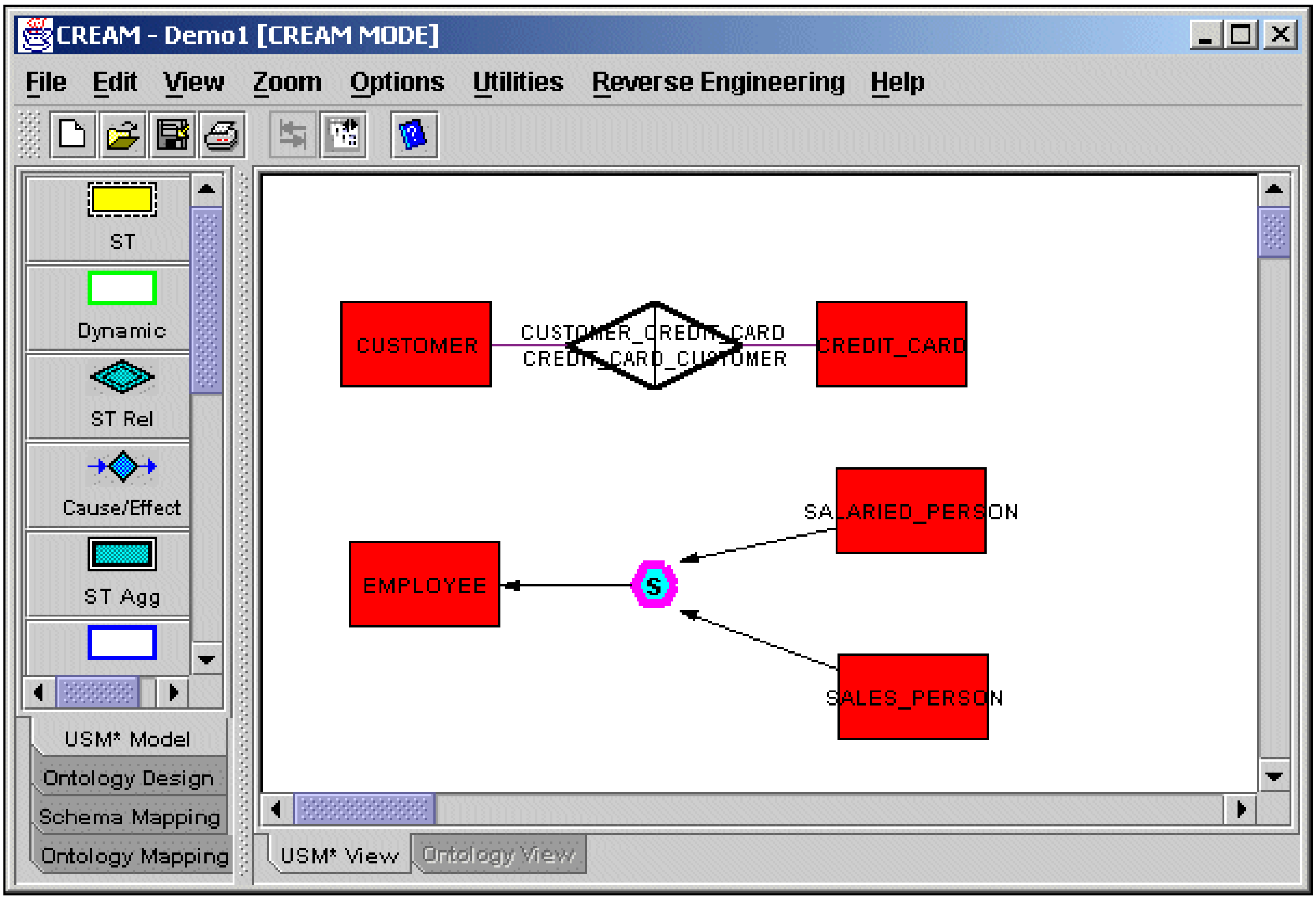
SALARIED_PERSON.EmpNo → EMPLOYEE.EmpNo

SALES_PERSON.EmpNo → EMPLOYEE.EmpNo

Figure 4: Sample Relational Schema: One-One & Super/Subclass

The primary key is underlined.

However, our agent does not know the exact cardinality constraints between the two relations. In order to detect minimum/maximum cardinality constraints, the agent initiates SQL queries to every instance of each relation's primary key attribute and stores minimum and maximum returned instance numbers from each query. In this example, the returned maximum and minimum number of instances for the 'CUSTOMER' relation are both one and the returned numbers for the 'CREDIT_CARD' relation are zero and one, respectively. Since both queries returned maximum numbers of one, our agent concludes that there exists a binary one-to-one (1:1) relationship. However, if the foreign key attribute (i.e., 'CustomerID') in the 'CREDIT_CARD' relation is constrained as a primary key attribute for the 'CREDIT_CARD' relation, the relationship must be treated as a super/subclass relationship, not a binary one-one relationship. Thus, the agent further checks on whether it is a super/subclass relationship. Since the 'CREDIT_CARD' relation only contains a 'CardNo' as primary key attribute, the relationship is concluded to be a binary one-one relationship. Figure 5 shows the reverse engineered output of this example and a dialog-box that illustrates detailed information on the reverse engineered relationship.



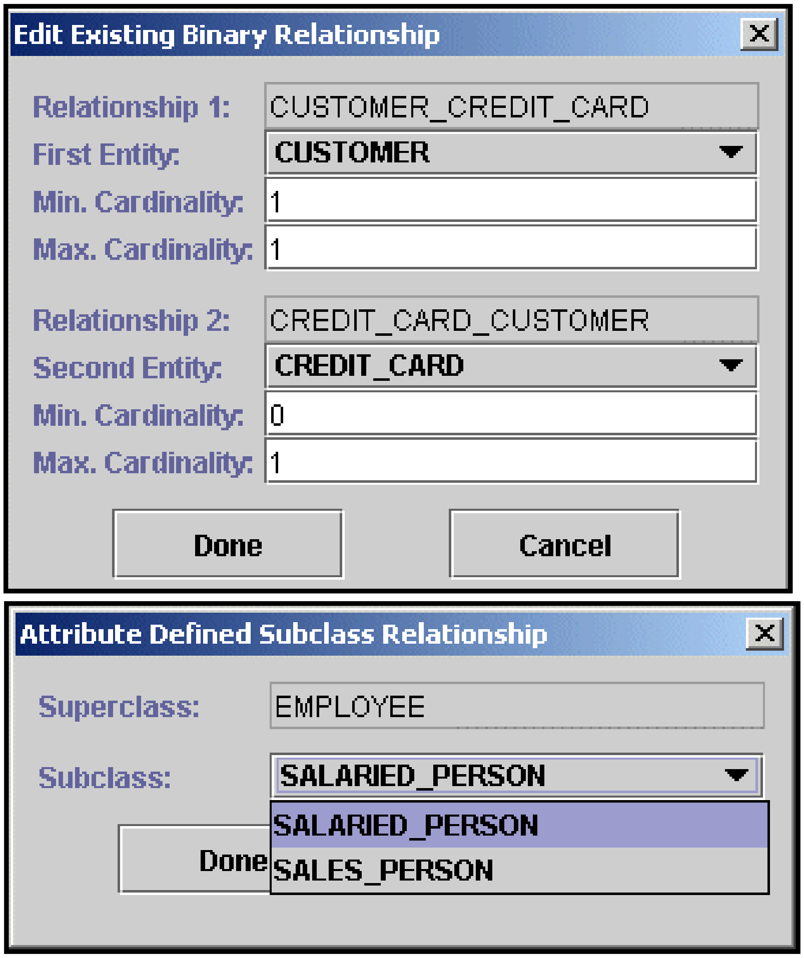


Figure 5: Result of the Reverse Engineering Process: One-One & Super/Subclass

After finishing the reverse engineering process for the first referential integrity constraint, our agent checks the next constraint, the 'SALARIED_PERSON' relation, which refers to the 'EMPLOYEE' relation. As no other relations refer to the 'SALARIED_PERSON' relation, the relationship between the relations is initially detected as a binary relationship. Our agent then extracts the minimum/maximum cardinality constraints between these two relations by initiating SQL queries for all instances of each relation's primary key attribute and stores the returned numbers of instances from the query. Since the returned maximum numbers of the two relations are one and one respectively, a binary one-one relationship is determined between the two relations.

Our agent then determines whether a given relationship is a super/subclass relationship by checking the 'SALARIED_PERSON' relation's primary key attribute(s) and foreign key attribute(s). If the foreign key attribute (i.e., 'EmpNo') of the relation is constrained as a primary key attribute, a super/subclass relationship is identified between the relations. Since the 'EmpNo' attribute of the 'SALARIED_PERSON' relation is constrained as a foreign key attribute in addition to the primary key attribute, the relationship between these two relations is determined to be a super/subclass relationship. Our agent then displays a dialog-box that allows the user to specify the type of super/ subclass relationship (e.g., 'Attribute-Defined' or 'Roster-Defined'). After a domain expert specifies the type, the agent creates the corresponding USM constructs and then proceeds to the next unfinished constraint.

The final constraint that is processed is the 'SALES_PERSON' relation, which refers to the 'EMPLOYEE' relation. Since no other relations contain the 'SALES_PERSON' relation, their relationship is initially detected as a binary relationship. Since the minimum/maximum cardinality constraints between the relations are both one, again a binary one-one relationship is determined. In this example, the 'SALES_PERSON' relation's foreign key attribute (i.e., 'EmpNo') is constrained as a primary key attribute for the 'SALES_PERSON' relation and the 'EMPLOYEE' relation contains an existing super/subclass relationship with the 'SALARIED_PERSON' relation. Therefore, it is concluded that the super/subclass relationship between the 'EMPLOYEE' relation and 'SALES_PERSON' relation is associated with the existing super/subclass relationship between the 'EMPLOYEE' relation and 'SALARIED_PERSON' relation.

The final results of the reverse engineering process for this example are illustrated in Figure 5. The dialog-boxes in Figure 5 shows detailed information for the binary one-one relationship between 'CUSTOMER' and 'CREDIT_CARD,' and the super/subclass relationship among 'EMPLOYEE,' 'SALARIED_PERSON,' and 'SALES_PERSON.'

Detection of Binary One-Many Relationship & Simple Weak Entity Class

One-Many Relationship	Foreign key in a relation refers to only one relation and maximum cardinality constraint between two relations is One-Many.
Simple Weak Entity Class	The relationship between two relations is One-Many cardinality and the primary key attributes in the child relation are composed of its own attribute(s) and the foreign key attribute(s).

Table 3: Reverse Engineering Rules: One-Many & Simple Weak Entity

As illustrated in Figure 6, four relations are stored in the RDBMS. The 'MEETING_ROOM' relation has its own attribute, 'RoomNo,' as well as the foreign key attribute, 'DeptNo,' which are both primary key attributes. The reverse engineering process of *SMEVA* begins with the first referential integrity constraint. Since the 'MEETING_ROOM' relation refers only to the 'DEPARTMENT' relation, the relationship between the two relations is initially detected as a binary relationship.

CUSTOMER (CustID, Name, Address, Phone, Email, Type)

ORDERS (OrderNo, OrderDate, DeliveryType, CustID)

DEPARTMENT (DeptNo, Name, Budget)

MEETING_ROOM (DeptNo, RoomNo, RoomName, Capacity)

Referential Integrity Constraints:

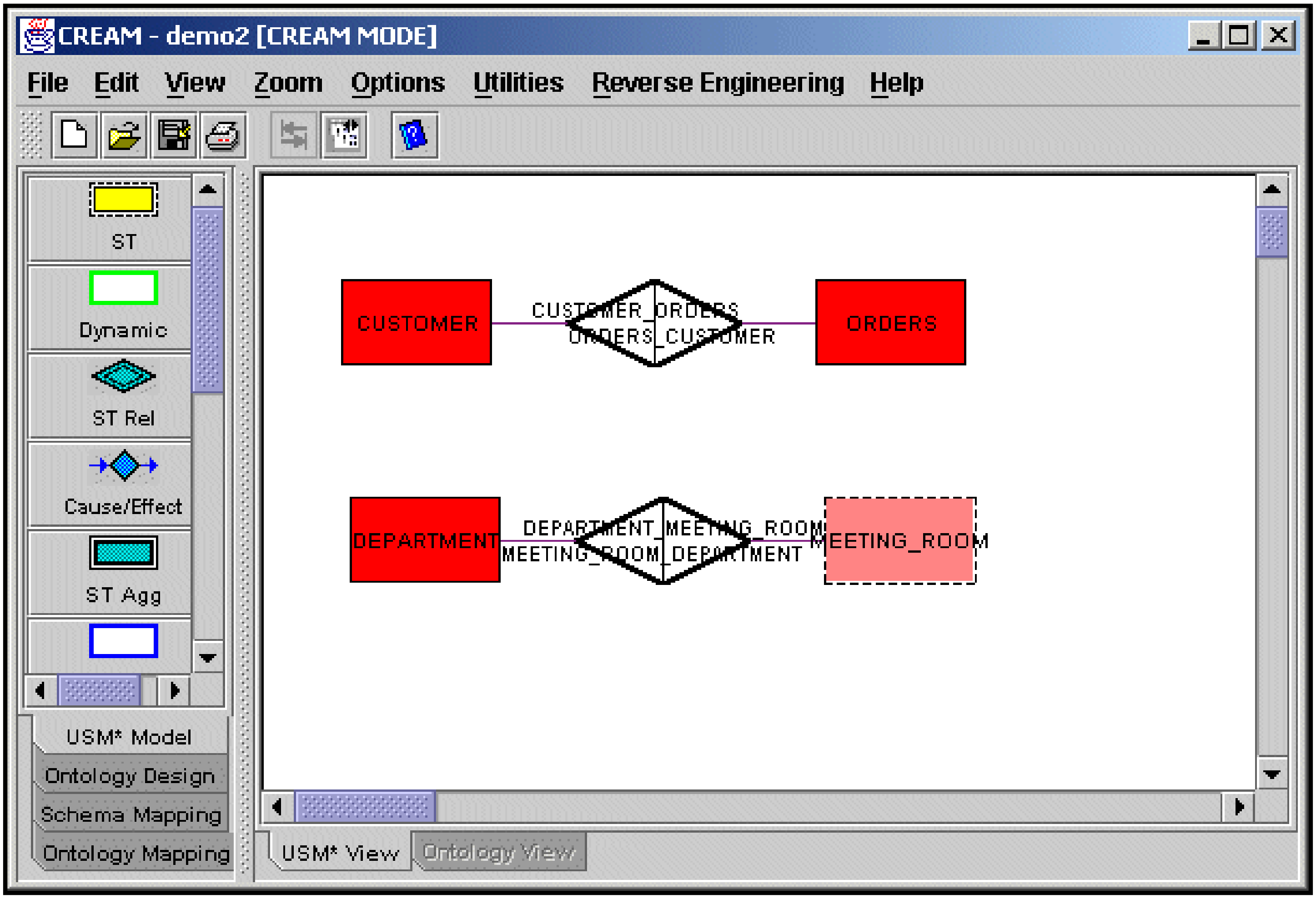
MEETING_ROOM.DeptNo → DEPARTMENT.DeptNo

ORDERS.CustID → CUSTOMER.CustID

Figure 6: Sample Relational Schema: One-Many & Simple Weak Entity
The primary key is underlined.

In order to determine the cardinality constraints between these two relations, our agent initiates an SQL query for every instance of each relation's primary key attribute and stores the minimum and maximum returned numbers from the query. In this example, the minimum/maximum returned instance for the 'DEPARTMENT' relation are both one, while the returned numbers for the 'MEETING_ROOM' relation are zero and many, respectively. As the maximum cardinality constraints for the two relations are one and many, respectively, a binary one-many(1:M) relationship is detected.

Before assigning this relationship between these two relations, our agent checks whether the relationship is a simple weak entity. If the foreign key attribute of the 'MEETING_ROOM' relation is constrained as primary key attribute of the 'MEETING_ROOM' relation, a simple weak entity relationship is determined. Otherwise, the relationship is concluded to be a binary one-many relationship. In this example, the foreign key attribute (i.e., 'DeptNo') of the 'MEETING_ROOM' relation is also constrained as a primary key attribute. Therefore, the relationship between the relations is determined to be a simple weak entity relationship (see Figure 7).



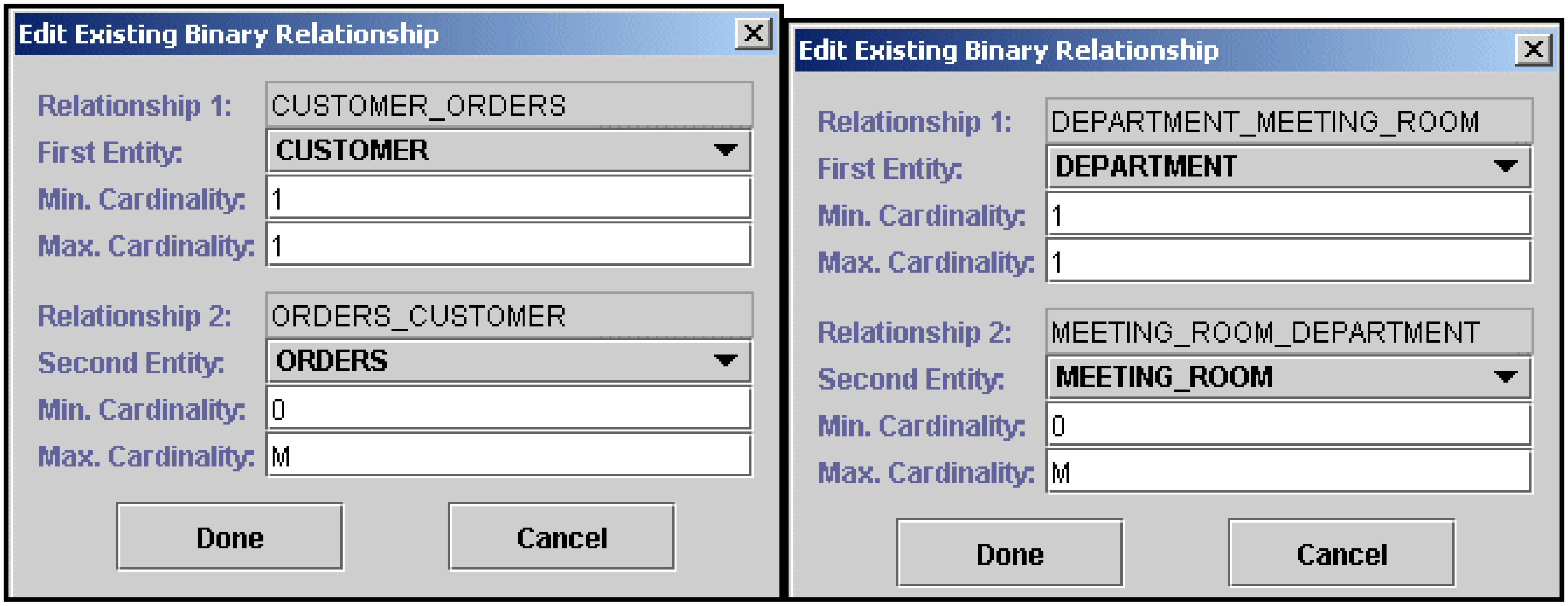


Figure 7: Result of the Reverse Engineering Process: One-Many & SimpleWeak Entity

Our agent then checks the next constraint (i.e., the 'ORDERS' relation refers to the 'CUSTOMER' relation). Since the 'ORDERS' relation refers to only the 'CUSTOMER' relation, the relationship between the two relations is considered a binary relationship. Next, our agent extracts the cardinality constraints between the relations by executing SQL queries. The minimum/maximum returned numbers for the 'CUSTOMER' relation are both one, while the numbers for the 'ORDERS' relation are zero and many, respectively. After identifying the correct cardinality constraints, the agent determines whether the relationship between the relations is a simple weak entity. Since the 'CustID' attribute, which is the foreign key attribute of the 'ORDERS' relation, is not constrained as primary key, the relationship between the 'CUSTOMER' and 'ORDERS' relation is concluded to be a binary one-many relationship. Figure 7 illustrates the final reverse engineered output and two dialog-boxes that show details of the two referential integrity constraints.

Detection of Binary Many-Many Relationship & Interaction Weak Entity Class

Many-Many Relationship	Foreign key in a relation refers to exactly two different relations and the relation has only foreign key attribute(s).
Interaction Weak Entity Class with Ternary Relationship	Foreign key in a relation refers to exactly two different relations and the relation has non- primary key or non-foreign key attributes.

Table 4: Reverse Engineering Rules: Many-Many & Interaction Weak Entity

The relational schema in Figure 8 shows typical examples of interaction weak entity class with a ternary relationship and binary many-many relationship. The 'PRICE_STRUCTURE' relation refers to both the 'CUST_TYPE' relation and the 'PRODUCT' relation. Since the 'PRICE_STRUCTURE' relation refers to two other relations, the relationship among the 'PRICE_STRUCTURE,' 'CUST_TYPE,' and 'PRODUCT' is initially detected as a binary many-many relationship.

Next, the minimum cardinality constraints between the 'CUST_TYPE' and 'PRICE_STRUCTURE' relations, and between the 'PRODUCT' and 'PRICE_STRUCTURE' relations are extracted by running an SQL query. Our agent checks whether the relationship is an interaction weak entity. To determine such a relationship, two conditions must hold. First, a foreign key attribute must refer to exactly two different relations. 'ProdID' refers to two relations 'PRODUCT' and 'PRODUCES,' so the first condition holds. Second, since the 'DiscountRate' attribute in the 'PRICE_STRUCTURE' relation is neither a foreign key nor a primary key attribute, it is concluded that the relationship is an interaction weak entity class with ternary relationship

DEPARTMENT (<u>DeptNo</u>, Name, Budget)

PRODUCES (DeptNo, ProdID)

PRODUCT (<u>ProdID</u>, ProdName, ProdPrice)

CUST_TYPE (<u>Code</u>, Name, Description)

PRICE_STRUCTURE (<u>CustType</u>, <u>ProdID</u>, DiscountRate)

<u>Referential Integrity Constraints:</u>

PRICE_STRUCTURE.CustType → CUST_TYPE.Code

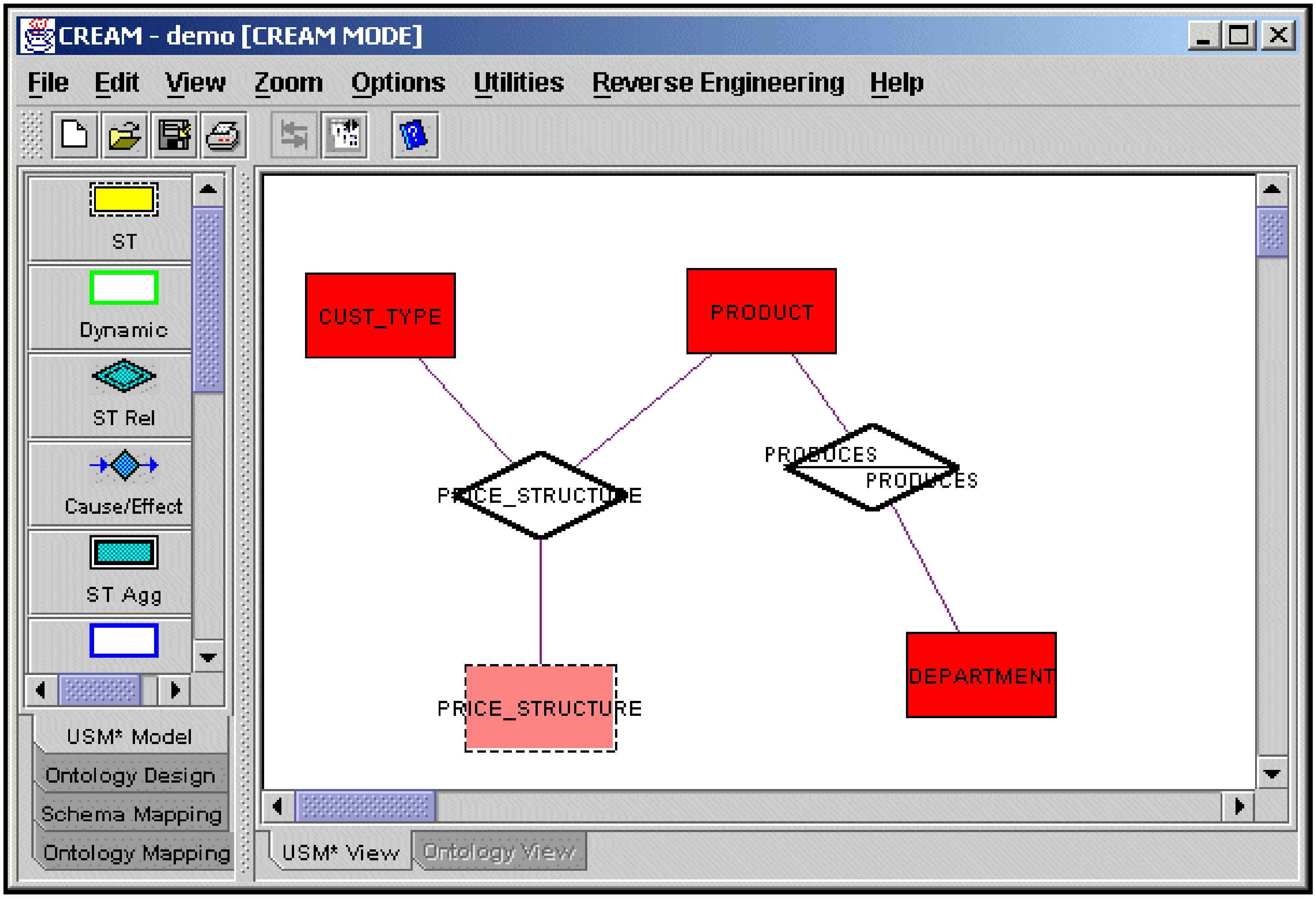
PRICE_STRUCTURE.ProdID → PRODUCT.ProdID

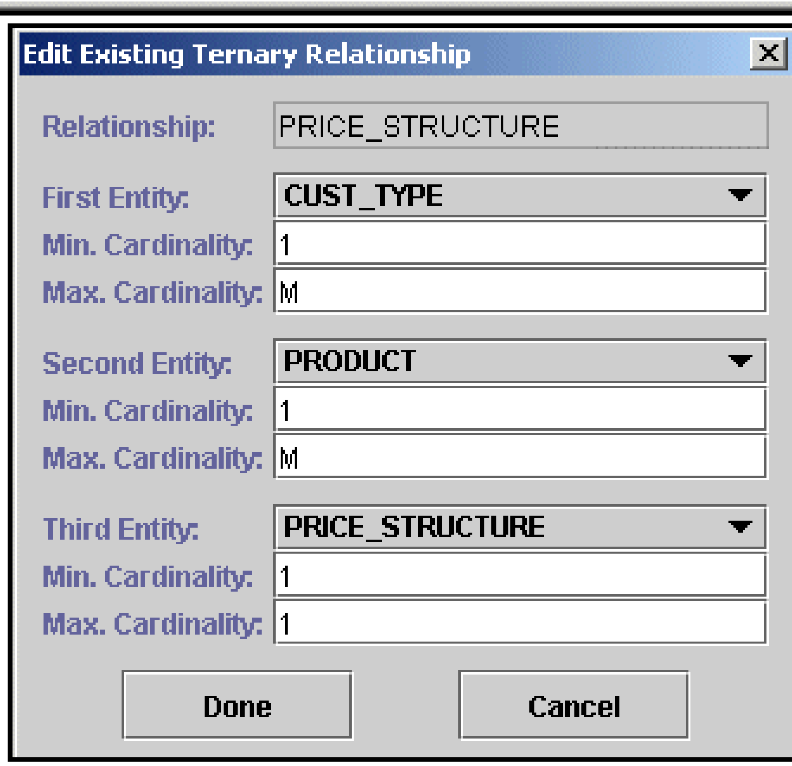
PRODUCES.DeptNo → DEPARTMENT.DeptNo

PRODUCES.ProdID → PRODUCT.ProdID

Figure 8: Sample Relational Schema: Many-Many & Interaction Weak Entity
The primary key is underlined.

Our agent continues to process the other referential integrity constraints, namely, the 'PRODUCES' relation, which refers to both the 'PRODUCT' and 'DEPARTMENT' relations. By counting the number of references to the 'PRODUCES' relation, the relationship among the 'PRODUCES,' 'DEPARTMENT,' and 'PRODUCT' relation is initially detected to be a binary many-many relationship.


CREAM - demo [CREAM MODE]
File Edit View Zoom Options Utilities Reverse Engineering Help
ST
Dynamic
ST Rel
Cause/Effect
ST Agg
USM* Model
Ontology Design
Schema Mapping
Ontology Mapping
CUST_TYPE
PRODUCT
PRICE_STRUCTURE
PRODUCES
PRODUCES
PRICE_STRUCTURE
DEPARTMENT
USM* View
Ontology View


Edit Existing Ternary Relationship
Relationship: PRICE_STRUCTURE
First Entity: CUST_TYPE
Min. Cardinality: 1
Max. Cardinality: M
Second Entity: PRODUCT
Min. Cardinality: 1
Max. Cardinality: M
Third Entity: PRICE_STRUCTURE
Min. Cardinality: 1
Max. Cardinality: 1
Done
Cancel

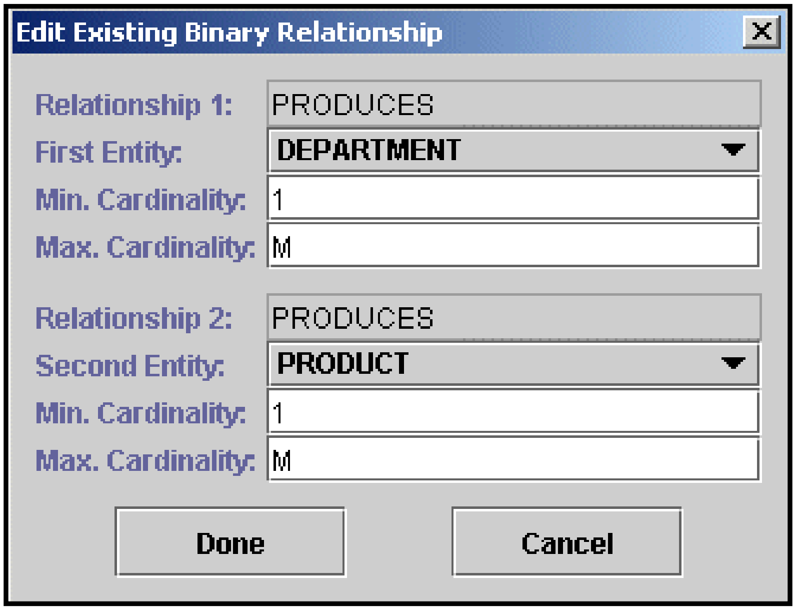


Figure 9: Result of the Reverse Engineering Process: Many-Many & Interaction Weak Entity.

Finally, our agent identifies the correct minimum cardinalities between 'PRODUCES' and 'DEPARTMENT' and between 'PRODUCES' and 'PRODUCT' by executing an SQL query. After identifying minimum cardinality constraints, our agent checks whether the relationship is an interaction weak entity class with ternary relationship. Since the 'PRODUCES' relation contains neither a foreign key nor a primary key attribute, and because the foreign key ProdID refers to exactly two relations, the relationship among the relations is determined as a binary many-many relationship. Figure 9 shows the reverse engineered output and two dialog-boxes that illustrate the detailed interaction weak entity class with ternary relationship and binary many-many relationship.

Detection of N-ary Relationship

N-ary Relationship	Foreign key in a relation refers to exactly N different relations.

Table 5: Reverse Engineering Rules: N-ary Relationship

When a given relation refers to three or more other relations, the relationship among the relations is determined to be an N-ary relationship. In the example shown in Figure 10, the 'SKILL_USED' relation refers to three other relations: 'EMPLOYEE,' 'PROJECT,' and 'SKILL.' Therefore, a ternary relationship among the four relations exists.

EMPLOYEE (EmpNo, SSN, Name)

PROJECT (ProjNo, ProjName, Budget)

SKILL (SkillCode, Name)

SKILL_USED (EmpNo, ProjNo, SkillCode)

Referential Integrity Constraints

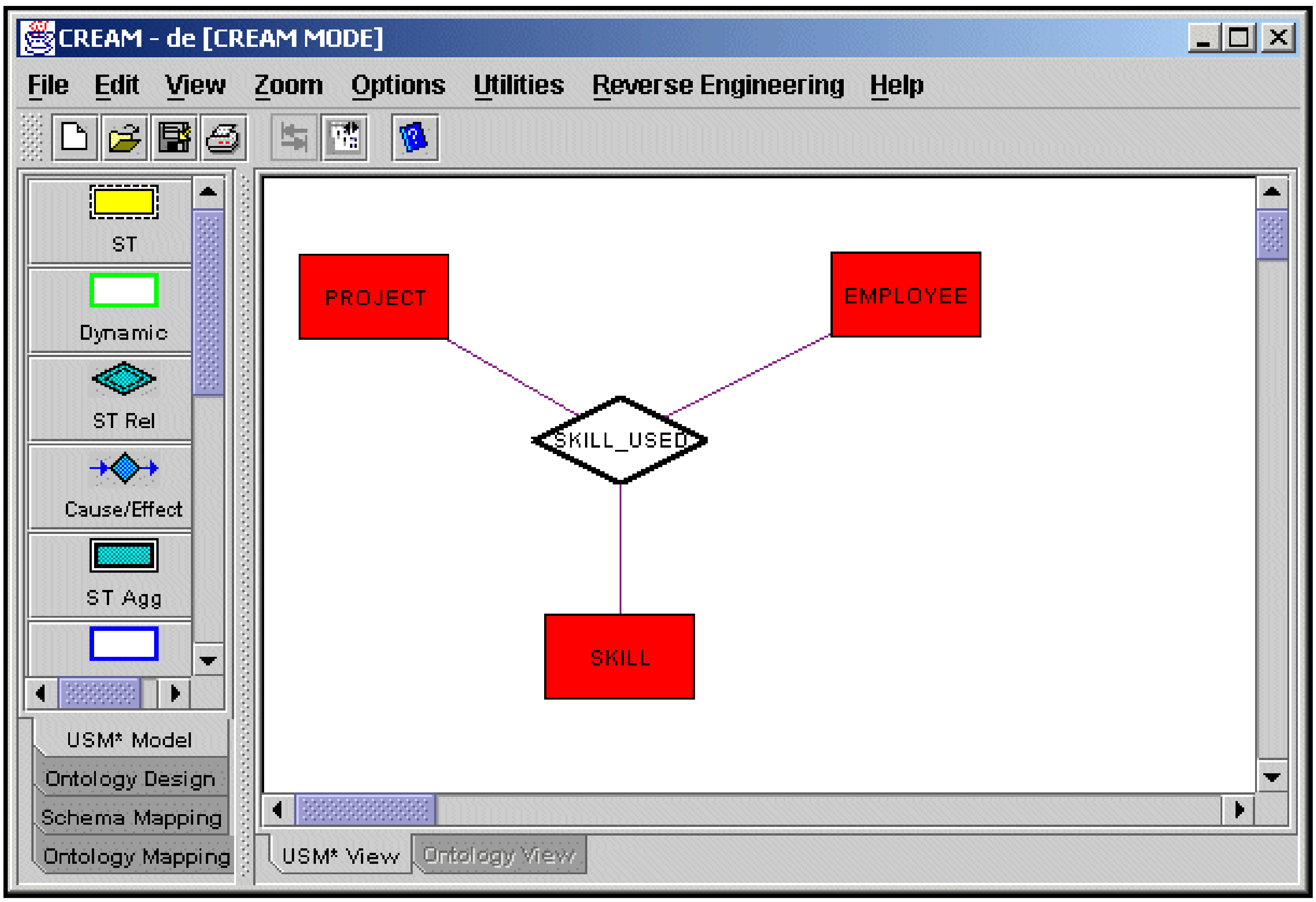
SKILL_USED.EmpNo → EMPLOYEE.EmpNo

SKILL_USED.ProjNo → PROJECT.ProjNo

SKILL_USED.SkillCode → SKILL.SkillCode

Figure 10: Sample Relational Schema: Ternary Relationship. ***The primary key is underlined.***

Our agent then identifies the minimum/maximum cardinality constraints of the relationship by executing SQL queries exhaustively. Finally, the USM construct for the ternary relationship is created as shown in Figure 11, which illustrates the output of this reverse engineering process. The dialog-box shows the exact cardinality constraints of the ternary relationship.



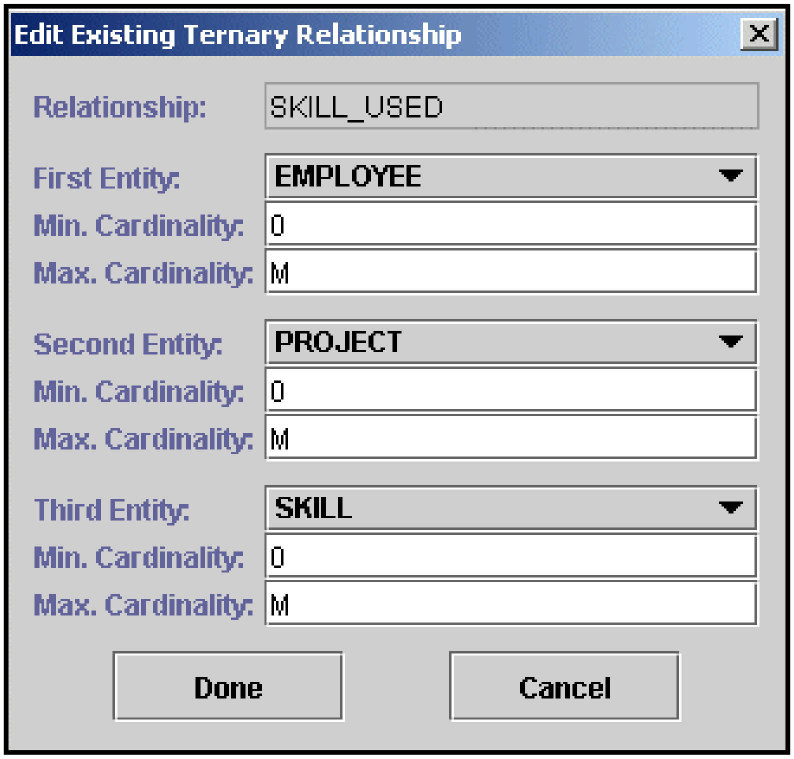


Figure 11: Result of the Reverse Engineering Process: Ternary Relationship

Verification and Modification of Resulting Schema

After transforming relations into relationships (one-one or one-many), *SMEVA* creates names of relationships by combining names of two associated entities. As the relationship name is usually in verbal form (e.g., 'takes' or 'taken_by'), this naming convention is not appropriate for normal conceptual schema design. However, our agent incorporates a modifiability function, which allows users to redefine names of relationships in such a manner to make the reverse engineered conceptual schema more intuitive.

Conclusion

In this paper, we implemented a semi-autonomous relational database reverse engineering agent, called *SMEVA*. The methodology presented extracts a conceptual model from an existing relational database by analyzing data instances as well as metadata (data schemas). *SMEVA* has several technological features distinguishing it from other DBRE-related research, including:

- *SMEVA* is coded with 100% pure Java programming language (JDK 1.3 including AWT and Swing) utilizing object-oriented design. One of the powerful features of the technology is platform independence. Thus, the agent runs on heterogeneous operating systems including MS Windows, UNIX, LINUX, Macintosh, etc. Another advantage of using Java is web accessibility. Several users can access the agent simultaneously through the web and utilize our agent to accomplish reverse engineering of any relational schema.

- *SMEVA* used the pure JDBC (Java Database Connectivity) API (v. 2.0). The JDBC API makes it easy to send SQL statements to relational database systems and supports all dialects of SQL. In addition, with the JDBC API, we could write a single program to access any RDBMS, such as Oracle, Sybase, IBM DB2, MS SQL Server, etc.

Limitations and Future Direction

Even database experts may occasionally violate rules of good database design and they often make use of unusual constructs [13]. Therefore, it may be impossible to generate a correct model using the database reverse engineering process. Although *SMEVA* generates a sound reverse-engineered model, it also has several limitations when used to reverse engineer an accurate conceptual model. Some limitations and possible extensions of our research are summarized as follows:

- Our heuristic algorithm to detect a super/subclass relationship is applicable when there is a one-one cardinality relationship between two relations and the primary key in a relation (subtype) is at the same time the foreign key of another relation (supertype). However, because our agent is not fully automated, our algorithm does not distinguish among different types of subclass relationships. Thus, our future work needs to refine the algorithm to fully support detecting the specific types of super/subclass relationship.

- Although the schema designer (i.e., USM) in the CREAM system [15] supports composite and aggregate/group relationships, our agent currently is unable to detect them. Future work is needed to develop a "*Knowledge Base*" that contains domain knowledge and several rules for detecting these relationships.

- Nowadays, an emerging method for delivering contents on the Web is eXtensible Markup Language (**XML**), which allows document authors to express semantic information in a standard way. Although semantic metadata delivered in an XML document is well-formatted in tree structure, it does not provide a good overview of its contents to end-users. On the other hand, semantic models such as USM can visualize all metadata information to end-users in more intuitive manner. Even though we definitely agree that XML is a good vehicle for delivering semantic metadata information, we still believe that semantic models can integrate XML in terms of visualizing the information. Within our research, the semantic knowledge extraction rules can be extended to incorporate XML documents into semantic model. Therefore, our next research will address application of the *SMEVA* agent's predefined rules to given XML documents.

Acknowledgements

We would like to give special thanks to *Dr. Sudha Ram* and *Dr. Jinsoo Park* for their valuable comments. Both of them encouraged us to write this article and carefully reviewed it.

Glossary

Schema:
: A schema defines the structure of the entire database and the type of contents that each data element within the structure can contain.

Relation Schema:
: A relation schema is used to describe a relation. A relation schema *R*, denoted by *R(A1, A2, A3,..., An)*, is made up of a relation name *R* and a list of attributes *A1, A2,..., An*.

Relational Schema:
: A set of relation schema. A description of multiple relational tables.

Conceptual schema:
: A conceptual schema describes the data items and relationships between data items in a form suitable for human presentation. An Entity-Relationship model is a form of a conceptual schema.

Primary key:
A primary key is an attribute selected to uniquely identify rows(tuples) within the relational table.

Foreign key:
A foriegn key is an attribute or set of attributes within one relational table that occurs as a primary key of another relational table.

Referential Integrity Constraint:
If a foreign key exists in a relation, either the foreign key value matches a primary key value of some tuple in its home relation or the foreign key value must be set to null.

Binary Relationship:
The degree of relationship is two; exactly two entities participate in a relationship.

Ternary Relationship:
The degree of relationship is three; exactly three entities participate in a relationship.

Quaternary Relationship:
The degree of relationship is four; exactly four entities participate in a relationship.

References

1
Batini, C., Seri, S., and Navathe, S. *Conceptual Database Design: An Entity Relationship Approach*. Benjamin Cummings, Redwood City, 1992.

2
Chen, P. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems* 1, 1 (Mar. 1976), pp. 9-36.

3
Chiang, R., Barron, T. and Storey, V. Reverse Engineering of Relational Databases: Extraction of an EER Model from a Relational Database. *Data & Knowledge Engineering* 12, 2 (March 1994), pp. 107-142.

4
Chiang, R., Barron, T., and Storey, V. (1996). A Framework for the Design and Evaluation of Reverse Engineering Methods for Relational Databases. *Data & Knowledge Engineering* 21, 1 (Oct. 1996), pp. 57-77.

5
Chikofsky, E., and Cross II, J. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software* 7, 1 (Jan 1990), pp. 13-17.

6
Davis, K., and Arora, A. Converting a Relational Database Model into an Entity-Relationship Model. In *Proceedings of the 6th International Conference on Entity-Relationship Approach* (Nov. 9-11, New York, NY), 1987, pp. 243-257.

7
Hammer, M., and McLeod, D. Database Description with SDM: A Semantic Database Model. *ACM Transactions on Database Systems* 6, 3 (Sep. 1981), pp. 351-386.

8
Johannesson, P. A Method for Transforming Relational Schemas into Conceptual Schemas. In *Proceedings of the 10th International Conference on Data Engineering* (Feb. 14-18, Houston, TX), 1994, pp. 190-201.

9
Markowitz, V. and Makowsky, J. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *IEEE transactions on Software Engineering* 16, 8 (Aug. 1990), pp. 777-790.

10
Navathe, S., and Awong, A. Abstracting Relational and Hierarchical Data with a Semantic Data Model. In

Proceedings of the 6th International Conference on the Entity-Relationship Approach (Nov. 9-11, New York, NY), 1987, pp. 277-305.

11

Park, J. *Facilitating Interoperability among Heterogeneous Geographic Database Systems.* Ph.D. Dissertation, Department of MIS, The University of Arizona, 1999.

12

Petit, J-M., Toumani, F., Boulicaut, J-F., and Koulomdjian, J. Towards the Reverse Engineering of Denormalized Relational Databases. In *Proceedings of the 12th International Conference on Data Engineering* (Feb. 26-Mar. 1, New Orleans, LA), 1996, pp. 218-227.

13

Premerlani, W., and Blaha, M. An Approach for Reverse Engineering of Relational Databases. *Communications of the ACM* 37, 5 (May, 1994), pp. 42-49.

14

Ram, S. Intelligent Database Design Using the Unifying Semantic Model. *Information and Management* 29, 4 (1995), pp. 191-206.

15

Ram, S., Park, J., Kim, K., and Hwang, Y. A Comprehensive Framework for Classifying Data- and Schema-Level Semantic Conflicts in Geographic and Non-Geographic Databases. In *Proceedings of the 9th International Workshop on Information Technology and Systems* (Dec. 11-12, Charlotte, NC), 1999, pp.185-190.

16

Soutou, C. Relational Database Reverse Engineering: Algorithms to Extract cardinality Constraints. *Data & Knowledge Engineering* 28, 2 (Nov. 1998), pp. 161-207.

17

Teorey, T., Yang, D., and Fry, J. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. *ACM Computing Surveys* 18, 2 (Jun. 1986), pp. 197-222.

Biography

Dongwon Lee is an MIS graduate student at the University of Arizona. He received a BBA degree in 1996 and an MBA degree in 1998 from Seoul National University, Korea. His research interests include semantic modeling, object-oriented and distributed database systems, agent-based eCommerce & digital library, and IT diffusion & adoption. He is a member of the Advanced Database Research Group at the University of Arizona. He can be reached at dlee@bpa.arizona.edu.

Yousub Hwang is a Ph.D. student of Management Information Systems (MIS) at the University of Arizona. His research interests are in semantic modeling, agent-based business intelligence, and geographic information systems. He received a BS degree in 1997 and an MS degree in 1999 from the University of Arizona. He is also a member of the Advanced Database Research Group at the University of Arizona. He can be reached at yhwang@bpa.arizona.edu.
