A Neutral Semantic Representation for Data Model and Schema Translation

S. Y. W. Su       S. C. Fang

Database Systems Research and Development Center, CSE#470
Department of Computer and Information Sciences
Department of Electrical Engineering
University of Florida, Gainesville, FL32611, USA

Technical Report Number: TR-93-023
July, 1993
A Neutral Semantic Representation for Data Model and Schema Translation*

S. Y. W. Su             S. C. Fang

Database Systems Research and Development Center, CSE#470
Department of Computer and Information Sciences
Department of Electrical Engineering
University of Florida, Gainesville, FL 32611
E-mail: su@pacer.cis.ufl.edu, sf@reef.cis.ufl.edu

Abstract

In order to achieve the interoperability of heterogeneous database systems, a semantics-preserving translation of the modeling constructs and constraints captured by different data models and defined in different schemata is a necessity. It is difficult to translate the constructs and constraints of one model directly into those of another model because 1) their terminologies and modeling constructs are often different, and 2) being high-level user-oriented models, their modeling constructs may have a lot of implied semantic properties which may or may not correspond to those of the others. If these high-level constructs and constraints are decomposed into some low-level, neutral, primitive semantic representations, then a system can be built to compare different sets of primitives in order to identify whether these constructs and constraints are identical, slightly different, or totally unrelated. Discrepancies among them can be explicitly specified by the primitive representations and be used in the application development to account for the missing semantics. In this paper, we present a neutral data model ORECOM which has been used in the development of a data model and schema translation system. The model is object-oriented and provides a small number of general structural constructs for representing the structural properties of high-level modeling constructs including those of object-oriented data models. It also provides a powerful knowledge rule specification language for defining those semantic properties not captured by the general structural constructs. The language is calculus-based and allows complex semantic constraints to be defined in terms of triggers and the alternative actions to be taken when some complex data conditions have or have not been satisfied. This paper also presents eight basic constraint types found in many semantically rich data models. These constraint types are represented in the neutral model by parameterized macros and their corresponding micro-rules. Parameterized macros are compact forms of neutral semantic representation which are used in an implemented system for comparing and translating the modeling constructs and constraints of different data models. The corresponding micro-rules are the detailed semantic descriptions of the constraint types. The translation of many modeling constructs and constraints found in several popular models into the neutral representation is illustrated by examples.

1. Introduction

In recent years, there has been much R&D work on heterogeneous database management systems (HDBMS) and their interoperability (e.g., see the proceedings of an NSF Workshop [NSF89], a collection of papers in this area of study [GUPT89], the review on multidatabase [BREI90], the special issue on heterogeneous databases [ACM90], and the discussion on the semantic issues in multidatabase systems [ACM91].) A HDBMS can consist of a number of either

* The initial support of this research was provided by the National Institute of Standards and Technology under grant# 60NANB7D0714 and the subsequent support has been provided by the Florida High Technology and Industry Council under grant# UPN92110316.
federated autonomous or tightly integrated component database systems. Their interoperability allows their data to be shared and exchanged. According to the three-layered structure of a HDBS proposed in [ELMA90], a HDBS must be able to perform the following functions to achieve the interoperability: (1) convert different data models and query languages, (2) translate and integrate schemata, and (3) process global as well as local transactions. The common problem underlying these functions is the data model inconsistencies which exist in these heterogeneous systems. This problem is called the domain mismatch problem in [BREI90] and the representational heterogeneity in [GANG91]. As an example, a primary-key is required for defining an entity in IDEF-1X model [LOOM86, 87], but is not required for an entity in EXPRESS [SCHE89, ISO92] nor for an object type in NIAM [VERH82]. In another case, a constraint supported in one data model may not be correspondingly supported by other models. For instance, the indirect mapping constraint on an interaction association type of OSAM* [SU89] is not available in the above three mentioned models. Also, a modeling construct in one model may appear to be identical to that of another except that some subtle difference in their associated constraints (e.g., different update rules associated with the referential constraint.) These semantic differences or discrepancies create great difficulties in data conversion, query translation, schema integration as well as transaction execution in heterogeneous systems. They need to be fully accounted for when data of one system are accessed by another system so that no semantic properties will be lost. Thus, the analysis of semantic properties underlying the existing modeling constructs and the identification of the semantic similarities and differences of these constructs are the fundamental problems that need to be solved before the true interoperability of heterogeneous database systems can be achieved.

Motivated by the above problems, we have studied a number of popular data models in an attempt to identify the underlying semantic properties of their modeling constructs and their associated constraints. The objective is to use their common properties and differences as a basis for the design and development of a neutral data model through which the modeling constructs and constraints of one model can be converted into those of another, and semantic discrepancies, if exist, can be identified and specified explicitly. These discrepancies can be used for the generation of explanations and be used by application developers, who use the converted schemata and databases, for incorporation into the new application systems so that no semantic loss will result. If these discrepancies can be specified in enough detail, they can also be used for automatic generation of program code which can be incorporated into the new application systems.

The neutral model described in this paper is the result of this research effort. It is an Object-oriented, Rule-based, Extensible Core Model (ORECOM) which provides a few very general structural constructs and a powerful rule specification facility for capturing fine semantic properties. The high-level modeling constructs of existing data models can then be decomposed and represented by ORECOM's primitive structural constructs and semantic rules. By comparing
their low-level neutral representations, the semantic discrepancies of high-level constructs can thus be identified and explicitly specified by semantic rules. In ORECOM, these rules are called **micro-rules**, which allows the specification of database operations (the trigger conditions) under which certain detailed database states need to be verified to determine the alternative database operations to be taken based on the result of verification. A calculus-based language is used as the rule specification language. For the convenience of expressing the semantic constraint types frequently found in data models and for avoiding the repeated specifications of these constraint types in terms of detailed micro-rules, we use parameterized **macros** to represent them. Each macro captures a generic constraint type (e.g., cardinality) and its variations are specified by the parameters of the macro. Thus, a macro corresponds to sets of micro-rules which define the constraint type and its variations. In this paper, examples of high-level modeling constructs taken from semantically rich data models such as IDEF-1X, NIAM, EXPRESS, and OSAM* are used to show how they can be mapped into the underlying ORECOM's macro and micro-rule representations through semantic decompositions. The applications of ORECOM as a neutral model for data model learning, schema translation, schema verification and optimization, and schema integration are also explained.

The remainder of the paper is organized as follows. Section 2 presents the neutral model ORECOM including its object-oriented structural constructs and its micro-rule specifications. Section 3 defines the eight basic semantic constraint types, which are frequently found in data models, and their macro and micro-rule representations. Section 4 provides some examples of decompositions of high-level modeling constructs into ORECOM representations. Other potential applications of ORECOM are also explained. Section 5 provides a conclusion.

2. ORECOM: a Neutral Model

In this section, we shall present the neutral model designed for inter-model and schema translation. First, we shall present the concept of semantic decomposition of modeling constructs and constraints. Second, the required characteristics of the neutral model are discussed. Then, ORECOM's facilities for modeling the structural and behavioral properties of a high-level data model (or an application) are described.

2.1 Semantic Decomposition

For the convenience of database users, all existing data models are designed in such a way that they provide a number of high-level structural constructs for defining the structural properties (attributes and relationships) of real world entities. Associated with these constructs, there are a number of constraints (keys, non-null, total participation, cardinality, etc.) which are either explicitly specified by some reserved keywords or implicitly specified in the structures for expressing various semantic restrictions that should be enforced by DBMSs. Due to the fact that these modeling constructs are high-level and user-oriented, they often carry a lot of semantics...
which may or may not be equivalent to those in the modeling constructs of another data model. For example, the association between two relations in the relational model is expressed by means of a cross reference between keys and foreign keys, and the referential constraint can be implemented in a DBMS using different deletion rules. Whereas, in the entity-relationship or ER model [CHEN76], the association is explicitly modeled by a relationship which implicitly uses a cascaded deletion for implementing the referential constraint. As another example, the generalization construct or superclass-subclass association between two classes in all object-oriented data models implies the property of inheritance yet their inheritance models can be rather different and thus have different implied object insertion and deletion behaviors. Due to these and many other types of differences, a direct translation between modeling constructs and constraints of different data models is not workable since many fine semantic distinctions can not be captured and there will be semantic losses or additions in the translation. In order to achieve a semantics-preserving translation, it is necessary to decompose a modeling construct and its associated constraints into some low-level structural and behavioral primitives which can then be used for comparing the modeling constructs and constraints of different models and be used to explicitly represent their discrepancies.

2.2 Required Characteristics of a Neutral Model

The existing user-oriented data models use different terminologies to name all things of interest in an application world (e.g., entities, objects, concepts, tuples, etc.) and the collections of these things (e.g., entity types, object types, classes, concept types, relations, etc.). They also use different terms to specify the structural relationships among these collections (e.g., attributes, instance variables, associations, relationships, bridges, links, etc.) In order to map the structural constructs to a neutral representation, the neutral model should adopt a general terminology to name things and their relationships. It should also provide a number of very primitive structural constructs so that the basic structural properties of high-level data models can all be represented by these primitives. Those semantic properties in these high-level constructs that are not captured by these structural primitives can be specified using a knowledge specification language. The separation of structural primitives from detailed semantic specifications using a rule language is important since the former provides a common structural representation and the latter can be used to explicitly state the different semantic properties associated with different high-level constructs.

In any database management system, the semantics of modeling constructs and constraints of a data model can be stated in terms of what the DBMS should do when data defined by these constructs and constraints are retrieved and manipulated. In other words, their semantic properties can be defined by the conditions under which certain actions need to be taken by the DBMS. For
example, part of the meaning of a key attribute is that, upon the insertion of an entity instance (or a tuple of a relation) or the update of a key attribute value, the DBMS needs to verify that its key attribute value is different from those of the other instances. As another example, part of the semantics of a superclass-subclass association is that, upon the deletion of a superclass object instance, the corresponding object instances in all its subclasses in the class hierarchy or lattice need to be deleted also. Thus, the semantics of data models can be defined in terms of the operational semantics of a DBMS rather than the semantics of words and languages that linguists and philosophers are interested in.

For the purpose of data model and schema translation, the neutral model should be "low-level" enough to allow the subtle semantic differences among data models to be distinguished, yet "high-level" enough so that the comparison between the neutral representations of the modeling constructs or constraints of any two data models can be easily carried out by a translation system.

In the neutral model to be described in the next section, we adopt the object-oriented paradigm for structural representation due to its generality and use a calculus-based knowledge rule specification language which uses triggers and object association patterns for the specification of operational semantics associated with modeling constructs and constraints.

2.3 ORECOM

ORECOM stands for an object-oriented, rule-based, and extensible core model. Its object orientation allows all things of interest (e.g., physical objects, abstract things, events, processes, functions, etc.) to be represented uniformly in terms of objects and object associations. Its rule specification facility allows complex semantic constraints found in the existing data models to be explicitly specified by knowledge rules with triggers. Its extensible feature allows new semantic properties to be introduced into ORECOM to account for the possible semantic extensions of the existing models or the introduction of new models. The extensible feature has been reported in [YASE91] and will not be addressed in this paper.

2.3.1. Structural Primitives

**Object.** Object is the atomic unit for modeling an application world. It can represent a physical entity, an abstract concept, an event, or anything of interest to an application. Two general types of objects are distinguished in ORECOM: self-naming objects and system-named objects. Self-naming objects are those identified by their values such as integer or real numbers, character strings, or some structures such as set, bag, list, or array of atomic data items used for defining other complex self-naming objects or system-named objects. System-named objects are those of interest to the application users such as employees, parts, projects, etc. They are uniquely identified by system-assigned object identifiers (OIDs) and are described in terms of self-naming
and/or other system-named objects. This distinction is commonly made in data models, e.g., the lexical and non-lexical objects in NIAM, and domain and entity objects in OSAM*.

**Association.** An association is a bi-directional link connecting two objects. It specifies that the pair of objects are structurally related. However, the specific semantic properties between them such as a system-named object's relationship with a self-naming object (or a domain value) through an attribute, a superclass object's relationship with its representation in a subclass, or a system-named object's relationship with another system-named object, etc., are not represented by the association. In ORECOM, the semantic properties of an association link are specified by micro-rules (to be described later). This separation of the general structural relationship from the specific semantic properties allows heterogeneous data modeling constructs to be mapped to the neutral structural representations and their semantic similarities and/or differences to be explicitly represented by micro-rules. An n-nary association (n > 2) is represented by n number of binary associations and the semantics of the n-nary relationship is again captured by rules.

An association in ORECOM is labeled by an association name as well as the direction of the association. For example, a company object (c) hires an employee object (e) is represented by "c *\text{Hires} e\text{*}" where the symbol "*" represents the association, "\text{*}" indicates the direction, and "Hires" is the association name. This association can also be equivalently represented as "c *\text{Works_for} e\text{*}" where "Works_for" is the inverse of "Hires". The name of an association is the same as an attribute in semantic data models or an object variable in object-oriented data models. Association names are important particularly when two objects are related by more than one association and each has its own semantics. For instance, "c *\text{Belongs_to} e\text{*}" and "c *\text{Hires} e\text{*}" are two associations with different meanings between the same pair of objects.

**Class.** An object class is an abstraction of or a type specification for a collection of object instances which share some common structural and behavioral properties. Object instances of a class are uniquely identified by instance identifiers (i.e., IIDs). Each IID is concatenation of a class identifier (CID) and an object identifier (OID). Using this identification scheme, the same real world object which can be identified in ORECOM by its OID can have many instances in different classes and can be distinguished by their IIDs. Corresponding to the distinction of object types made in many existing data models, ORECOM categorizes object classes into two general types: entity-class and domain-class. An entity-class (or E-class) defines the structural and behavioral properties of a set of system-named objects. It also serves as the "container" of "holder" of a set of object instances which are the data representations of the collection of objects in that class. In ORECOM, an E_class is defined in terms of a class name, a number of associations with other classes, a set of method specifications (or signatures) and their implementations for defining the procedural semantics implemented in program code, and a set of micro-rules for defining semantic constraints applicable to its objects. A skeletal class definition of an Employee class is given in
Figure 1. A domain-class (or D-class) in ORECOM defines a set of self-naming objects (e.g., simple self-naming objects such as integers, reals, etc., or complex self-naming objects such as a list of integers, an array of reals, etc.) It specifies the data type and, optionally, some constraint(s) of a set of simple or structured values which are used for representing the instances of E-class objects. The values of a D-class are not explicitly entered and stored in a database. They are contained in the instances of E-class objects. In Figure 1, Eid, E_Salary, and Works_for are class association names which connect the E-class Employee to D-classes Integer and Salary and to another E-class Company, respectively. An instance of Employee consists of an Eid value, an E_Salary value, and an instance reference to a company object. We note that the traditional concepts of "attributes" and "entity associations/relationships" are uniformly represented in structure by "associations" in ORECOM. Similarly, the relationship between a superclass and a subclass recognized in object-oriented models (the same relationship is called a generalization or categorization in some semantic models), say, between class Person and class Employee, is represented by a class association linking Person to Employee. As we pointed out before, the detailed semantic properties of different association types such as "attribute", "entity association", "superclass/subclass or generalization association" are captured in ORECOM by micro-rules.

When two object classes are associated with each other as defined in a schema, their objects and thus object instances may also be associated with each other through the same association. For
example, the class association "Company *\rightarrow Hires Employee" implies that there can be an object association "c *\rightarrow Hires e" for some c in class Company and some e in class Employee.

Based on the primitive structural constructs (objects and object associations, classes and class associations) described above, the modeling constructs of many high-level data models can be represented in ORECOM's neutral representation. For example, Figure 2 shows the graphical representations of the constructs used in EXPRESS, IDEF-1X, NIAM, OSAM*, SDM [Hamm81, Thom89], and OMT [Rumb91] for modeling the superclass-subclass relationship between Employee and Secretary. Although, these data models use different terminologies and graphical symbols to represent their constructs, the underlying common structural properties can be specified by the concepts of E-classes Employee and Secretary and their class association. Additional to the structural primitives, a set of semantic constraints representing the implied meaning of superclass-subclass relationship can be explicitly defined by a set of methods and
micro-rules which correspond to the constraint statements shown in the figure for capturing the behavioral properties of these two classes and their association. By translating high-level modeling constructs into ORECOM's neutral structural primitives and detailed micro-rules, subtle differences among these constructs can be uncovered. For example, in IDEF-1X, it is required to specify a "discriminator" for the superclass-subclass relationship between Employee and Secretary. The discriminator is one attribute of the superclass, whose value determines which subclass a superclass object should be an instance of. This requirement does not exist in the other models mentioned above. It can be defined in terms of the insertion behavior followed by a DBMS.

2.3.2 Behavioral Primitives

The traditional data models such as relational, network and hierarchical models provide some structural constructs and very limited constraint specification facilities (e.g., by keywords such as Keys, Non-Null, domain constraints, etc.) They do not capture the behavioral properties of objects like object-oriented models do in terms of operations or methods which represent the procedural semantics of objects. In the traditional DBMSs, this type of semantic properties are either implemented in application programs or hard-coded in DBMSs. More recent data models such as the ones used in ODE [AGRA89], OSAM* [SU89], STARBUST [LOHM91], POSTGRES [STON91], etc., provide high-level rule specification facilities for defining different types of semantic constraints. Thus, in order to accommodate these existing data models and their translations, a neutral data model like ORECOM needs to provide the facilities for method and rule specification.

Method Specification. Like in other object-oriented data models, each system-defined or user-defined object class in ORECOM may contain a number of method specifications each of which defines the operation that can be performed on its members. Each method specification has a name, a number of arguments associated with the operation and optionally a returned value (e.g., the DisplayData and SetSalary in Figure 1.) The activation of an operation on an object is carried out by message passing. Methods can be either system-defined or user-defined. System-defined methods are object operations that are common to all class objects and are pre-defined in system-defined classes and inherited by user-defined classes. User-defined methods are object operations that are application-specific and applicable only to the objects of user-defined classes, in which these operations are specified, and their subclasses. Corresponding to each method specification, there is a method implementation containing the detailed program code that carries out the operation.

Since user-defined methods are application dependent and the procedural semantics captured by them have to be expressed by program code in a data model or schema translation, we shall present the system-defined methods of ORECOM in the remainder of this section. We shall
use the dot notation "x.op" to specify that the object operation op is performed on the object x. There are five system-defined object operations supported in ORECOM: CREATE, DESTROY, ASSOCIATE, DISASSOCIATE, and READ. These operations are corresponding to the structural primitives of objects and object associations discussed before. The operation \( x.\text{CREATE} \) establishes the object x in a class where the operation is executed. Its inverse operation \( x.\text{DESTROY} \) terminates the membership of x in that class. For establishing an object association between two objects x and y, the associate operation \( x.\text{ASSOCIATE}(\alpha, y) \) is used. Here, \( \alpha \) specifies the name of the association. The inverse of ASSOCIATE is DISASSOCIATE as in \( x.\text{DISASSOCIATE}(\alpha, y) \). We note here that, since many types of relationships found in the existing data models are modeled uniformly in ORECOM as objects and class associations, data manipulation operations such as update, insert, and delete are modeled by ASSOCIATE and DISASSOCIATE operations in ORECOM. For example, updating an attribute value of an object can be represented by disassociating the object with the old value (a D-class object) and associating the object with the new value. Inserting an object instance is represented by creating the object and associating the object with a number of D-class and/or E-class objects through different association names (i.e., attributes). The inverse operations can be done for the deletion operation. The last system-defined operation, READ, which is represented as \( x.\alpha \), is used for specifying the retrieval of all objects that are associated with x through the association \( \alpha \).

**Rule Specification.** Constraints captured by high-level data models are used by DBMSs to control the data retrieval and manipulation operations so that these constraints are enforced and the database integrity is maintained. They can be explicitly defined in terms of knowledge rules which specify the operational behaviors of objects and their associations. In ORECOM, these knowledge rules are called "micro-rules" since they are used to specify the detailed semantic properties of high-level data modeling constructs and constraints.

The rule specification language used in ORECOM for defining micro-rules is the rule specification part of a general-purpose knowledge base programming language called K which has been designed and implemented at the Database Systems Research and Development Center of the University of Florida [SHYY92, ARRO92]. The language is a calculus-based language and has triggers and association pattern specification capabilities. The syntax of a micro-rule is given below:

```
rule rule_id is
  triggered trigger_conditions
  [condition guarded_expression]
  [action statements]
  [otherwise statements]
```
The rule_id is a unique name for rule identification purpose. Besides the rule_id, a rule consists of a set of trigger_conditions and a rule body defined by condition, action, and otherwise clauses. When any of the trigger_conditions is satisfied, the rule is triggered and the rule body is evaluated. A trigger_condition is defined by a triggering time (i.e., before or after) and a system-defined or user-defined method. For example, the triggering condition "after x.ASSOCIATE(α, y)" states that after establishing an association named α between objects x and y, the following rule body should be evaluated. The condition clause can be specified by a guarded expression in the form of "G1, ..., Gn | T", where Gi, i = 1..n, are the guards and T is the target. The G's and T are Boolean expressions and may contain association patterns (to be explained below). A guarded expression is evaluated to the values of TRUE, SKIP, or FALSE. The value TRUE is returned if G1, ..., Gn, and T are all true. In this case, the statements specified in the action-clause will be executed. If any one of G1, ..., Gn is false in a sequential evaluation of these expressions, the guarded expression returns the value SKIP, which will cause the rest of the rule body to be ignored. If all the guards are true but the target T is false, then the expression returns FALSE. A false result will cause the statements specified in the otherwise-clause to be executed. The guarded expression is a short hand for a complex expression that involves the nesting of many condition-action-otherwise sub-expressions. The statements in both action-clause and otherwise-clause can be expressions of various kinds including system and user-defined methods or any K's computation statements including assignments, quantified expressions, conditional statements, repetitive statements, and so on [SHYY91, 92].

Besides the usual predicates used in Boolean expressions, the rule specification language of ORECOM allows "patterns of object associations" or simply "association patterns" to be specified in the condition clause. For example, the expression "x:X[PX]" is a simple association pattern which specifies that all objects of class X which satisfy the predicate expression PX and can be referenced by the object variable x. Thus, the expression "e:Employee[Age > 50 AND Salary = 70K]" would identify all Employee objects whose age is greater than 50 and whose salary is equal to 70K. The variable 'e' ranges over this set of employee objects. A more complex association pattern may involve two classes in the form of (x:X *α y:Y) or (x:X !α y:Y). (Here, predicates PX and PY associated with classes X and Y are omitted to keep the expressions simpler.) The pattern (x:X *α y:Y) returns all pairs of X and Y objects which are associated (as specified by the association operator "*" ) through α, whereas the pattern (x:X !α y:Y) returns all X objects which are not associated (as specified by the non-association operator "!") with any Y objects through α and all Y objects which are not associated with any X objects through the same association. The object variables 'x' and 'y' are used to reference the objects that satisfy the
expressions. As discussed before, the direction symbols, ">" and "<", distinguish the subject/agent from the object of an association and the following conditions hold:

"x:X *> α y:Y" = "x:X *< α⁻¹ y:Y" and
"x:X !> α y:Y" = "x:X !< α⁻¹ y:Y"

where α⁻¹ is the inverse association of α.

Association patterns may involve a long linear tree or network structure of object classes. They may also contain AND/OR branches and loops. They provide a simpler way of specifying complex associations among object classes and thus their objects. We shall explain the branching structures: "x:X AND(L1 P1, L2 P2, ..., Ln Pn)" and "x:X OR(L1 P1, L2 P2, ..., Ln Pn)". The Li in these expressions specifies an association link such as "*> αi" or "!< αi" and Pi represents a linear sub-tree or sub-network structured association pattern, i = 1..n. In a branching association pattern with a logical AND, the set of objects returned from class X must associate or not associate (depending on each Li specification) with every pattern of Pi, i = 1..n. In a branching association pattern with logical OR, those returned X objects must associate (or not associate) with at least one pattern of Pi, i = 1..n.

The syntax and semantics of micro-rules and the association patterns explained above provide a powerful means for specifying a variety of semantic constraints found in the existing data models. We have used such a rule specification language to define all the constructs and constraints of several semantically rich data models such as EXPRESS, IDEF-IX, NIAM, and OSAM*.

3. General Constraint Types and Their Micro-rule Representations

In this section, we shall present a number of semantic constraint types commonly found in the existing modeling constructs. They are the results of analyzing and relating the constructs and constraints of a number of semantically rich data models. For each constraint type, we introduce a "macro" representation, which is a parameterized way of specifying the constraint type and its variations. Corresponding to each variation, a set of micro-rules can be defined to specify its detailed operational semantics. Thus, a variation of a macro is an abstraction of a set of detailed micro-rules. The macro representation is particularly suitable for the comparison and conversion of modeling constructs and constraints since it is a more compact representation than the micro-rule representation. The former can be used by a data model and schema translation system to compare the ORECOM representations of modeling constructs and constraints and the latter can be used by the system to generate explanations or program code to account for the discrepancies found in a translation.

Our analysis of the modeling constructs and constraints of data models follows the following approach. We first determine what are the constraint types and their variations that are meaningful to a set of objects in an object class (i.e., intra-class constraints). We then examine
the possible constraint types and variations that are meaningful to the association of two object classes (i.e., the semantics of a binary association or inter-class constraints). Lastly, we examine those constraint types and variations related to an object class and its multiple associations with other object classes (i.e., the semantic of n-ary associations or inter-association constraints). This approach provides a systematic way of determining the semantic properties that can exist in the structural primitives of ORECOM.

3.1 Intra-class Constraints

In spite of the differences in terminologies and notations, all existing data models provide some ways of specifying the structures (or data types) of a set of objects that form a class (or its equivalent concepts), the ways objects can be individually identified, the constraints associated with the membership of objects in the class. Figure 3 provides some examples. In the IDEF-1X notation shown in 3(a), Eid and Ename are used together as a required external identifier (or primary key) of Employee objects. In 3(b), the data type Positive defined in EXPRESS allows only positive non-zero integers to be its members. Figure 3(c) shows that the attribute ProjTeam of the entity class Company in OSAM* is defined over the domain class EmpSet having a constraint of five to twenty employee instances in each set which in turn is defined over the entity class Employee. In Figure 3(d), each member of the NIAM's so-called lexical object type (LOT) Date is assumed to represent a tuple of objects from other LOTs Month, Day, and Year which are not explicitly shown in the schema.

In ORECOM, we introduce a constraint type or macro called MEMBERSHIP (MB), which has the following syntax (a parameterized notation) and semantics (in the form of a micro-rule):

Macro MB (X, O, I, S, T, C, M)  
Micro-rule rule MB-01 is /* defined in the meta-class named CLASS whose instances are
The above MB macro specifies properties and constraints of an object class that must be satisfied by its members. These properties and constraints are specified by a class name (X), an object type (O) which takes the value of "self-naming" or "system-named", a user-defined object identifier (I) (i.e., attribute(s) that serves as a primary key) if the object type is system-named, an object structure (S) which specifies the structure of its members (simple or complex structures such as SET, LIST, BAG, ARRAY, TUPLE, etc.), a class type (T) which is either an E-class or a D-class, a set of membership constraints (C) which either list the possible members, or specify the range of values that constrain its members, or express in logical expressions that evaluate to true or false, and a set of methods (M) which specify the meaningful operations that can be performed on the class members. A type definition in a high-level data model can be mapped into the MB macro with specific values assigned to its parameters. For example, the macro representations of the four classes shown in Figure 3 are as follows:

MB(Employee, system-named, (Eid, Ename), simple, E-CLASS, -, - )
MB(Positive, self-naming, -, simple, Integer, Positive > 0, - )
MB(EmpSet, self-naming, -, SET, Employee, -, - )
MB(Date, self-naming, -, TUPLE, (Month, Day, Year), -, - )

The "-" sign in the above macros indicates that a parameter is not specified with respect to the original construct, and the "E-CLASS" of the first macro is a system-defined class of ORECOM which contains all entity classes that have been defined. By translating the constructs of different data models which capture the concept and constraints of MEMBERSHIP into the above uniform macro representations and comparing their parameter values, a schema translation system will be able to identify their semantic similarities and differences.

The system-defined meta-class CLASS is a class containing all definitions of classes in terms of the following seven attributes: ClassName, ObjectType, Identifier, Structure, ClassType, Constraint, and LocalOperation. The micro-rule MB-01 shown above simply sets the attribute
values for a class, which are specified in a MB macro, before the class is created as an object instance (identified by "this") in the class CLASS.

### 3.2 Inter-Class and Inter-Association Constraints

We now examine some of the constraint types, which are commonly seen in the existing data models, for restricting the association between two object classes (inter-class constraints) and the multiple associations of an object class (inter-association constraints).

#### 3.2.1 PARTICIPATION (PT)

As an inter-class constraint, this constraint type restricts the total number of objects of one class that *must* participate in an association with the objects of another class. It is a very general constraint type which can be used as a common representation of a number of constraints used in different data models. Figure 4 shows some schema examples taken from different data models.

<table>
<thead>
<tr>
<th>S1 (IDEF-1X)</th>
<th>S2 (EXPRESS)</th>
<th>S3 (SDM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>ENTITY Company; Fax : OPTIONAL INTEGER; Phone: OPTIONAL INTEGER; ..... WHERE Fax &lt; 9999999999; Fax &lt;&gt; Phone; END_ENTITY;</td>
<td>CLASS Engineer (Position : INTEGER, INITIALVALUE 10; Salary : REAL, DERIVED BY Position * 325.3 + 20000; ) ;</td>
</tr>
<tr>
<td>SSN (AK)</td>
<td>S4 (OSAM*)</td>
<td>S5 (NIAM)</td>
</tr>
<tr>
<td></td>
<td>S6 (OMT)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4. Examples of schemata of different data models.

The alternate key, SSN, of Employee entity in the IDEF-1X schema S1 is also a non-null attribute. It requires that every Employee entity must be associated with a social security number. In other words, there is a total participation constraint associated with the Employees' associations with social security numbers. In the schema S2 defined in EXPRESS, both Fax and Phone are optional attributes. In ORECOM's representation, this is a partial participation constraint associated with
the Company class's associations with Fax class and Phone class since company objects may, but
do not have to have Fax and Phone numbers. In the SDM model, an attribute can have an initial
value assigned to it. The schema S3 shows that the initial value for the Position of each Engineer
object is '10'. This default attribute value assignment implies the constraint of total participation
since an engineer will have a position value equal to either 10 or some other number but not null.
The schema S4 shows an interaction association (I) defined in the OSAM* model to capture the
semantics that each Works_for object represents the fact that an Employee object is associated with
a Project object and the association itself is modeled as a Works_for object. This construct, among
other semantic properties, has a total participation constraint, namely all Works_for objects have to
be associated with some Employee objects as well as with some Project objects. Existence
dependency is another frequently used inter-class constraint, which is available in both the NIAM
schema S5 and the OMT schema S6 in Figure 4. In S5, every Full_Time_Student object and every
Part_Time_Student object must also be a Student object, that is, they are all existence dependent on
the Student objects. From ORECOM's point of view, both Full_Time_Student and
Part_Time_Student classes are totally participated in their associations with the Student class. (The
symbol "T" in S5 specifies a total specialization which means that a student object must be in either
Full_Time_Student class or Part_Time_Student class. This total specialization is an example of an
inter-association constraint of the PARTICIPATION constraint type which will be described later.
The other symbol "X" in S5 specifies a set-exclusive constraint which means that an object in
Full_Time_Student class can not be in Part_Time_Student class and vice versa. The set-exclusive
constraint belongs to another inter-association constraint type called Logical-Dependence which
will be addressed in Section 3.2.7.) The Car class in S6 is an aggregation of classes Engine,
Body, and Wheels, and according to the semantics of aggregation defined in the OMT model
[RUMB91], a Car object can not exist if any of its aggregation components (i.e., an Engine object,
a Body object, and a Wheels object) does not exist. In this case, there are three total participation
constraints on the Car class, one for each of these three associations. On the other hand, not every
Engine (or Body, or Wheels) object is used in a car, that is, there is only a partial participation
constraint associated with Engine's (or Body's, or Wheels') association with Car. The above
examples clearly show that the modeling constructs of different data models may look very
different. However, if their semantics are decomposed into more primitive representations, part of
their semantics may overlap and can be explicitly identified (in the above examples, the common
semantics are the variations of the PARTICIPATION constraint type).

We shall now consider the more general representation of the PARTICIPATION constraint
type and its macro and micro-rule representations.

Macro: \( PT (X, \text{min}, \text{max}, \alpha, Y) \) (M2.1)
Micro:

Rule PT-01 is /* defined in class X */
triggered after this.CREATE, immediate_after this.DISASSOCIATE (α,y)
condition exist x in ( x:X *α Y where Count(x) < min )
action REJECT;

Rule PT-02 is /* defined in class X */
triggered immediate_after this.ASSOCIATE (α,y)
condition exist x in ( x:X *α Y where Count(x) > max )
action REJECT;

The inter-class PARTICIPATION (PT) macro uses two parameters, min and max, to specify a lower-bound and an upper-bound of the number of X objects which participate in the association named α with the class Y. The values of min and max can be zero, a positive integer, or any expression that returns a positive integer. The min value is always less than the max value. If min equals to zero, it means that there is no constraint on the lower-bound. If it equals to the expression "Count(X)", which returns the current total number of objects in class X, then a total participation constraint is specified. On the other hand, if max equals to zero, it means that no object can participate in the association. Max equals to "Count(X)" simply means that all X objects can participate. As an example, the macro "PT (Employee, 5, Count(Employee), SSN, Integer)" specifies that at least five employees must have social security numbers, whereas the macro "PT (Employee, Count(Employee), Count(Employee), SSN, Integer)" specifies a total participation which states that every employee must have a social security number.

Using the macro defined in (M2.1), the macro representations of some variations of the PARTICIPATION constraint type which exist in the modeling constructs of Figure 4 are shown below:

S1: PT(Employee, Count(Employee), Count(Employee), SSN, Integer) --- a non-null attribute
    PT(Integer, 0, Count(Integer), SSN-1, Employee) --- a partial participation
S2: PT(Company, 0, Count(Company), Fax, Integer) --- an optional attribute
S3: PT(Engineer, Count(Engineer), Count(Engineer), Position, Integer)
    --- a default attribute value which implies a non-null constraint
S4: PT(Works_for, Count(Works_for), Count(Works_for), -, Employee)
    --- an implied total participation
S5: PT(Full_Time_Student, Count(Full_Time_Student), Count(Full_Time_Student), -, Student)
    PT(Part_Time_Student, Count(Part_Time_Student), Count(Part_Time_Student), -, Student)
    --- existence dependencies
S6: PT(Car, Count(Car), Count(Car), -, Engine) --- an existence dependency
    PT(Engine, 0, Count(Engine), -, Car) --- a partial participation

The micro-rule PT-01 is an example operational rule which can be defined in class X to enforce the participation constraint of X objects. It specifies the enforcement of the constraint after
a CREATE transaction has been performed on X or after a DISASSOCIATE operation is executed on an object of X identified by "this" and an object of Y identified by "y". The function "Count(x)" returns the number of those X objects which fall in the association pattern "x:X *α Y". This value is compared with the min value. Here, x is an object variable which ranges over those X objects that satisfy the pattern. "After" means that the rule is verified after the CREATE transaction instead of right after the CREATE operation which is specified using the key-word "immediate_after". In the CREATE transaction, the association between the created X object and some Y object(s) can be established. If the minimum participation constraint is violated after the creation transaction or after the disassociation operation, the operation will be rejected. The rejection of the creation transaction will cause the transaction to be rolled back and the rejection of the disassociation operation will cause it to be aborted. The rule PT-02 ensures that the upper-bound max is not violated by an ASSOCIATE operation by comparing it with the current number of the participating X objects (i.e., Count(x)).

Sometimes it is desired that a participation constraint applies only to a subset of objects of a class rather than the entire set of objects as in the examples shown above. For instance, in OSAM*, a user-defined constraint can be stated as a rule inside the Employee class such as "all employees who have no Eids must have SSNs". This is a total participation constraint for only a subset of employees rather than all employees. The condition that all employees who have no Eids is called a selection condition of the Employee class. The incorporation of a selection condition to each class specified in the participation macro makes it even more general and useful for capturing some user-defined constraints. For this reason, we extend the participation macro M2.1 into the following form:

Macro: $PT(X, P_X, min, max, \alpha, Y, P_Y)$

The semantics of this macro then becomes that at least min and at most max X objects which satisfy the selection condition $P_X$ must participate in the association named $\alpha$ with those Y objects which satisfy the selection condition $P_Y$. The macro defined in M2.1 can be viewed as a special case of M2.2 with both selection conditions omitted. Using M2.2, the above OSAM* example can be represented as "PT(Employee, Employee.Eid = nil, Count(Employee), Count(Employee), SSN, Integer, - )" where no selection condition is specified for the Integer class. Accordingly, micro-rules PT-01 and PT-02 can be modified to include the expressions $P_X$ and $P_Y$. Additional rules must also be introduced to account for those operations that may change the qualification of some X or Y objects with respect to $P_X$ and $P_Y$.

A participation constraint can also exist in a class which has multiple associations with other classes. For example, as shown in the NIAM schema S5 of Figure 4, Student class has a total specialization constraint (identified by the symbol "T") in its associations with
Full_Time_Student and Part_Time_Student classes, meaning that every student must be in either one of the subclasses. Or, in terms of the participation concept, Student objects must be totally participated in some associations with some objects of these two subclasses (or any number of subclasses in a more general case). Thus, the participation macro is further generalized in the following form:

Macro:  \( \text{PT} \left( X, P_X, \min, \max, (\alpha_1, Y_1, P_{Y_1}), ..., (\alpha_n, Y_n, P_{Y_n}) \right) \)  

This macro states that the participation of those \( X \) objects that satisfy \( P_X \) and are in the associations \( \alpha_1, \alpha_2, ..., \alpha_n \) with those \( Y_1, Y_2, ..., Y_n \) objects that satisfy \( P_{Y_1}, P_{Y_2}, ..., P_{Y_n} \), respectively, must be within the range of \([\min, \max]\). It is of no importance which association and how many associations a qualified \( X \) object actually participates in. Using the macro M2.3, the total specialization constraint of the NIAM schema S5 can be represented in ORECOM by "\( \text{PT} \left( \text{Student}, -, \text{Count} \left( \text{Student} \right), \text{Count} \left( \text{Student} \right), (-, \text{Full\_Time\_Student}, -), (-, \text{Part\_Time\_Student}, -) \right) \)". in which no selection conditions are specified for the three classes in this example.

3.2.2 CARDINALITY (CD)

The CARDINALITY constraint type specifies that the number of objects of one class which can be associated with an object of another class must be in a specified range. It is a general constraint type that can exist in an association between two classes as well as among associations of multiple classes. In the inter-class case, a cardinality constraint is commonly expressed as follows:

\[
X : Y = [\minX, \maxX] : [\minY, \maxY]
\]

This expression means that an \( X \) object can associate with a minimum number of \( Y \) objects specified by \( \minY \) and a maximum number of \( Y \) objects specified by \( \maxY \), and that a \( Y \) object can be associated with a minimum number of \( X \) objects specified by \( \minX \) and a maximum number of \( X \) objects specified by \( \maxX \). Using this format, we can define some cardinality constraints for the schemata given in Figure 4 as below (the Ms in the expressions mean "many"):

S1: Employee : SSN = [1,1] : [1,1] -- employee's SSN is single-valued and unique
S2: Company : Fax = [1,M] : [1,1] -- company's FAX is single-valued and non-unique
S3: Engineer : Salary = [1,M] : [1,1] -- engineer's salary is single-valued and non-unique
S4: Employee : Works_for = [1,1] : [1,M] -- each Works_for object involves one employee but more than one Works_for object can be associated with an employee
S5: Student : Full_Time_Student = [1,1] : [1,1] -- a student can only have one representation as a full time student and vice versa.
S6: Car : Engine = [1,1] : [1,1] -- one car, one engine and vice versa

The general representation of the inter-class CARDINALITY constraint type is defined by the following macro and micro-rules.
Macro: \( \text{CD} (X, \alpha, Y, \text{min}X, \text{max}X, \text{min}Y, \text{max}Y) \) \text{ (M3.1)}

Micro:

Rule \textbf{CD-01} is /* defined in class \( X \) */
triggered before this.DISASSOCIATE(\( \alpha, y \))
condition exist \( y' \) in ( this \( \rightarrow \alpha y':Y \) where Count(\( y' \)) = \( \text{min}Y \))
action REJECT

Rule \textbf{CD-02} is /* defined in class \( X \) */
triggered before this.ASSOCIATE(\( \alpha, y \))
condition exist \( y' \) in ( this \( \rightarrow \alpha y':Y \) where Count(\( y' \)) = \( \text{max}Y \))
action REJECT

Rule \textbf{CD-03} is /* defined in class \( Y \) */
triggered before this.DISASSOCIATE(\( \alpha^{-1}, x \))
condition exist \( x' \) in ( this \( \leftarrow \alpha x':X \) where Count(\( x' \)) = \( \text{min}X \))
action REJECT

Rule \textbf{CD-04} is /* defined in class \( Y \) */
triggered before this.ASSOCIATE(\( \alpha^{-1}, x \))
condition exist \( x' \) in ( this \( \leftarrow \alpha x':X \) where Count(\( x' \)) = \( \text{max}X \))
action REJECT

The values of the four parameters in the CD macro, i.e., \( \text{min}X, \text{max}X, \text{min}Y, \) and \( \text{max}Y \), can be any positive integers or the special character 'M'. For example, the value \([1, M]\) of \([\text{min}Y, \text{max}Y]\) means that an \( X \) object can associate with many \( Y \) objects, whereas the value \([3, M]\) limits the minimum number of \( Y \) objects that can be associated with an \( X \) object to three. In the latter case, the non-zero \( \text{min}Y \) does not imply that every \( X \) object must be associated with some \( Y \) object. But, if an \( X \) object is associated with some \( Y \) object, it must also be associated with two other \( Y \) objects to satisfy the minimum cardinality constraint. Using M3.1, the cardinality constraints of the above examples can be represented in the neutral macro forms:

S1: \( \text{CD(Employee, SSN, Integer, 1, 1, 1, 1)} \)
S2: \( \text{CD(Company, Fax, Integer, 1, M, 1, 1)} \)
S3: \( \text{CD(Engineer, Salary, Real, 1, M, 1, 1)} \)
S4: \( \text{CD(Employee, -, Works_for, 1, 1, 1, M)} \)
S5: \( \text{CD(Student, -, Full_Time_Student, 1, 1, 1, 1)} \)
S6: \( \text{CD(Car, -, Engine, 1, 1, 1, 1)} \)

To enforce the cardinality constraint specified in a CD macro, four micro-rules named CD-01, CD-02, CD-03, and CD-04 are defined as shown above to maintain the four bounds. The lower-bound \( \text{min}Y \) can be violated only by a DISASSOCIATE operation which removes the association of an \( X \) object with a \( Y \) object and hence may decrease the number of the associated \( Y \)
objects to a value less than minY. Rule CD-01 checks if the number of Y objects associated with
the X object is already equal to minY. If so, the disassociation operation is rejected. In this rule,
the function Count(y') gives the total number of Y objects that satisfy the pattern "this *\alpha \ y':Y",
where 'this' names the X object operated on by the triggering DISASSOCIATE operation. The
interpretations of CD-02, CD-03, and CD-04 are similar to CD-01.

Similar to the extension of the PARTICIPATION macro, the CD macro in M3.1 can be
extended as below by incorporating selection conditions to allow a cardinality constraint to be
applied on selected subsets of objects.

Macro:  CD (X, P_X, \alpha, Y, P_Y, minX, maxX, minY, maxY)
(M3.2)

In the above, we have presented a CD macro and micro-rules for representing a cardinality
constraint on a single association between two classes. The same constraint type may exist among
multiple associations of a class. We shall use the SDM schema S3 of Figure 4 as an example. In
that schema, an additional cardinality constraint may state that a pair of Position and Salary values
may be associated with a minimum of one and a maximum of six Engineer objects. Here, the
cardinality constraint is added between the Engineer class and the pair of Integer and Real classes
through the two associations Position and Salary. In the following, we shall introduce two
generalized CD macros for two cases of inter-association cardinality constraints. The first one is a
macro for the cardinality constraint between one class and a set of directly associated classes as
shown in Figure 5(a), and the second one is for the cardinality constraint between two indirectly
associated sets of classes as shown in Figure 5(b).

The macro representation of the first case is given as M3.3 below. This macro specifies
two bounds, minY and maxY, which stand for the minimum and maximum numbers of tuples of
qualified Y1, Y2, ..., Yn objects that a qualified X object can be associated with (through the
associations \alpha1, \alpha2, ..., and \alpha n, respectively). It also specifies the minimum and maximum
numbers of qualified X objects with which a tuple of qualified Y1, Y2, ..., and Yn objects can be
associated with through the associations \beta1, \beta2, ..., and \beta m, respectively.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.png}
\caption{Two inter-association cardinality constraints:}
(a) X : (Y1, ..., Yn) = [minX, maxX] : [minY, maxY]
(b) (Z1, ..., Zm) : (Y1, ..., Yn) = [minZ, maxZ] : [minY, maxY]
\end{figure}
Macro:
\[
\text{CD } (X, \ P_X, ((\alpha_1, \ Y_1, \ P_{Y_1}), ..., (\alpha_n, \ Y_n, \ P_{Y_n})), \ \text{minX}, \ \text{maxX}, \ \text{minY}, \ \text{maxY})
\]  
(M3.3)

Using M3.3, our previous example of inter-association cardinality constraint can be represented as "CD(Engineer, -, ((Position, Integer, -), (Salary, Real, -)), 1, 6, 1, 1)" with no selection conditions specified.

The micro-rule representation of M3.3 is basically the same as that of M3.2 except that additional rules of the forms similar to CD-03 and CD-04 are needed because of the multiple classes Y1, Y2, ..., and Yn, and the Count function of CD-01 and CD-02 is extended to count the number of associated tuples of Y1 ... Yn objects associated with an X object.

The second case of a cardinality constraint is illustrated in Figure 5(b), where two sets of classes, (Z1, ..., Zm) and (Y1, ..., Yn), are indirectly associated through another set of classes (X1, ..., Xk). A cardinality constraint can be specified for the two sets of associations or attributes, (\beta_1, ..., \beta_m) and (\alpha_1, ..., \alpha_n), such that 

\[
(Z_1, ..., Z_m) : (Y_1, ..., Y_n) = [\text{minZ}, \text{maxZ}] : [\text{minY}, \text{maxY}].
\]

The macro that represents such an inter-association cardinality constraint is defined as follows:

Macro:  
\[
\text{CD } (((\beta_1, \ Z_1, \ P_{Z_1}), ..., (\beta_m, \ Z_m, \ P_{Z_m})), ((X_1, \ P_{X_1}), ..., (X_k, \ P_{X_k})), ((\alpha_1, \ Y_1, \ P_{Y_1}), ..., (\alpha_n, \ Y_n, \ P_{Y_n})), \ \text{minZ}, \ \text{maxZ}, \ \text{minY}, \ \text{maxY})
\]  
(M3.4)

This macro is the most general form of all cardinality constraints. In other words, all the previous presented CD macros, including M3.1, M3.2, and M3.3, are simply special cases of M3.4. An example of this general constraint type is found in OSAM* model. The indirect cardinality constraint in the OSAM* schema S4 of Figure 4 can be represented in M3.4 form with the indices m = k = n = 1. In this schema, the Employee and Project classes are indirectly associated through the Works_for class and their mapping relationship is specified to be [3, 15] : [1, 2], meaning an employee can work for at most two projects and a project can have at least three and at most fifteen employees. This constraint can be translated, using M3.4, into its macro representation "CD((- , Employee, -), (Works_for, -), (- , Project, -), 3, 15, 1, 2)" in which no attribute names and no selection conditions are specified. An extension of the micro-rules of M3.1 for the general macro M3.4 is straight-forward using the rule specification language.

### 3.2.3 INHERITANCE (IH)

Inheritance is a modeling mechanism which allows object attributes, operations, and rules of a superclass to be inherited by objects of a subclass. It is a very useful and commonly supported modeling construct available in most new generation of semantic and object-oriented data models. However, in more traditional data models such as the relational model and the E-R model, this
construct is not supported. To accommodate both types of data models, ORECOM does not treat the inheritance as one of its basic structural constructs. Instead, it is treated as a basic constraint type whose semantics is expressed by rules. The following IH macro and its micro-rule illustrate one aspect of the operational semantics of inheritance.

Macro: \textbf{IH (X, Y)} \hfill (M4.1)

Micro:
\begin{itemize}
  \item \textbf{rule IH-01} is /* defined in class Y (the subclass) */
  \item \textbf{triggered} before this.att, this.DISASSOCIATE(att, v), this.ASSOCIATE(att, v), this.op
  \item \textbf{condition} (not \textbf{In}(att, this.LocalAttribute)) \lor (not \textbf{In}(op, this.LocalOperation)) \mid
  \text{exist x in } ((this \rightarrow x:X) \text{ where } \text{OID}(this) = \text{OID}(x))
  \item \textbf{action} \quad x \$ this.thisOperation;
\end{itemize}

The IH macro has two parameters: X specifies a superclass, and Y specifies a subclass. The six inheritance constructs of EXPRESS, IDEF-1X, NIAM, OSAM*, SDM, and OMT shown in Figure 2 can be uniformly represented in ORECOM as "IH(Employee, Secretary)". The IH macro is an inter-class constraint type. If a superclass has more than one subclass, then each superclass-subclass association will be represented by an IH macro. Similarly, in the case of multiple inheritance, there will be one IH macro for each association between the subclass and one of its superclasses. Other constraints between the superclass and the subclass (e.g., existence dependency, cardinality, or the discriminator specification as required by the IDEF-1X model) and among the multiple superclass-subclass associations (e.g., total specialization, set exclusion, set subset, etc.) are represented by other types of macros.

The semantics of one aspect of inheritance is described by micro-rule IH-01, which is defined in the subclass of the association identified by Y. Generally speaking, this rule allows an operation defined in a superclass to be performed on objects of a subclass. The four triggering operations of the rule IH-01 including attribute retrieval, attribute association, attribute disassociation, and activation of a user-defined operation. In these triggering specifications, 'this' represents a Y object, 'att' and 'v' represent an attribute and its value, and 'op' represents a user-defined operation. A prerequisite condition for evaluating the rule body is that an attempt is made to access or manipulate an attribute (att) or to perform an operation (op). The condition-clause of IH-01 is a guarded expression which states that, if the attribute being retrieved or manipulated or the operation being performed is not a member of the set of attributes or operations defined in class Y, then we want to check if there exists an object instance in class X that corresponds to "this" Y instance having the same OID. If both conditions verified in sequence are true, then the triggering operation is aborted to be performed on the corresponding X object. This is done by using a
The casting operator "$" which replaces the original operand, a Y object instance (this), with its corresponding superclass object instance (x) to avoid a type checking error.

The general concept of inheritance should apply to not only the inheritance of attributes (or associations) and operations but also the inheritance of rules. However, it is noted that the definition of IH-01 dose not show the rule inheritance. This is because that references to the inherited attributes and operations in a subclass are replaced by references of these attributes and operations defined in its superclass. Processing of these inherited attributes and operations are thus automatically subject to the semantic rules of the superclass. In other words, the function of rule inheritance is achieved by the object casting mechanism.

In the following, a more general IH macro is given to allow selection conditions to be used with both superclass and subclass. To our knowledge, none of the existing models provide an explicit way to model the inheritance property for some selected objects. However, such restriction could have been introduced in a model or defined by user-defined rules.

Macro: \textbf{IH} (X, P_X, Y, P_Y) \hspace{1cm} (M4.2)

3.2.4 \textbf{PRIVACY} (PV)

Constraints of this category provide access protections to the structural and behavioral properties of objects. An association between two classes can be declared to be private, protected, or public. For example, suppose α is an attribute defined in class X, whose domain is class Y. If this attribute (or association between X and Y) is private, then only objects of X can initiate operations to access the Y objects that are associated with the X objects through α or to associate or disassociate with Y objects. This type of privacy constraint can also be applied to a superclass-subclass construct. For example, if X is a subclass of Y, only X objects can initiate an inherited operation or access an inherited attribute from class Y and Y’s superclasses. In both modeling constructs, the association is private or "invisible" to all other objects associated with class X. For a protected association, on the other hand, the privilege of traversing the association to access Y objects and its attributes and operations is also granted to objects of the subclasses of X. In this case, if class Z is a subclass of X and X is a protected subclass of Y, then objects of Y and its superclasses and their operations and attributes will be all inheritable to both X and Z objects. Similarly, if Y is the domain of the protected attribute α of X, and z represents a subclass object of Z, then the retrieve operation "z.α" will be granted. An association which is neither private nor protected is accessible to all objects and is called a public association.

Privacy constraints are not commonly supported by the existing data models. However, they are supported by some object-oriented programming languages such as C++. The two simple C++ schemata shown in Figure 6 give some examples of privacy constraints. In schema S7, the Employee class has a public attribute Eid, a protected attribute Address, and a private attribute
Salary. In schema S8, the superclass-subclass association between the Employee and Manager classes is defined to be a private association. Because of this privacy constraint, the inheritance of Employee's instances, attributes, and operations is limited to objects of Manager only. They can neither be further inherited by subclasses of Manager nor be accessible to Manager's other associated classes. The C++ privacy constraints have been adopted in the underlying data model of our implemented knowledge base programming language K.1 [ARRO92].

---

Figure 6. Examples of public, protected, and private associations in C++.

Macro: \texttt{PV( X, \alpha, Y, privacy\_type )} \hspace{1cm} (M5.1)

Micro:

rule \texttt{PV-01} is /* defined in class X */

triggered before this.\alpha, this.ASSOCIATE(\alpha, y), this.DISASSOCIATE(\alpha, y),

\hspace{1cm} y $ this.thisOperation

condition ((privacy\_type = "private") => (this.prior = this)) OR

\hspace{1cm} ((privacy\_type = "protected") =>

\hspace{3cm} (((this.prior = this)) OR ((this.prior *$ this)) AND (OID(this.prior) = OID(this))))

otherwise \texttt{REJECT;}  

The PV macro shown in M5.1 is a general form to represent the above privacy constraints. The same macro can represent a privacy constraint of a superclass-subclass association or a simple association between two classes. The value of the parameter 'privacy\_type' can be specified as "private" or "protected". Using this PV macro, the privacy constraints in S7 and S8 of Figure 6 are represented as follows:

S7: \texttt{PV(Employee, Salary, float, private)}

S7: \texttt{PV(Employee, Address, char, protected)}

S8: \texttt{PV(Manager, -, Employee, private)}

The PV macro is enforced by the micro-rule PV-01. This rule is defined in class X and can be triggered by two sets of operations, depending on whether there is inheritance on the association. If there is no inheritance, then PV-01 can be triggered by one of the three operations:
this.$\alpha$, this.ASSOCIATE($\alpha$, y), or this.DISASSOCIATE($\alpha$, y). If Y is a superclass of X, then it can be triggered by a casting operation which transfers an operation from class X to class Y. The main task of PV-01 is to find out, through the use of a system-defined method called prior, the object which initiates the triggering operation. It can be the current operated X object identified by 'this', or objects of other classes. For the former case, the method "this.prior" will return a value which is the instance identifier (IID) of the object represented by 'this'. The returned value must equal to 'this' for a private association, because only objects of X can initiate the triggering operations. However, for a protected association, the value of "this.prior" can be equal to 'this' or the IID of a subclass instance of 'this'. In our last example shown in Figure 6, we can assume Secretary is another subclass of Employee and let 's001' and 'e001' represent the IIDs of the person in the Secretary and Employee classes, respectively. Then, the operation "s001.Salary" will be replaced with another operation "e001 $ s001.Salary" and before it is executed, the rule PV-01 in Employee class is triggered. In evaluating this PV-01, the value returned by the method "e001.prior" is s001, not e001. According to the private constraint which requires "e001.prior" to be equal to e001, the operation "e001 $ s001.Salary" and hence the original "s001.Salary" will thus be rejected. On the other hand, the retrieve operation "s001.Address" will be accepted since Address is a protected attribute of Employee and can be accessed by the subclass objects of Employee (e.g., s001).

The following macro is a generalized PV macro of M5.1, which allows selection conditions to be specified with both classes.

Macro: PV( X, P_X, $\alpha$, Y, P_Y, privacy_type )

(M5.2)

3.2.5 TRANSITION (TS)

This type of constraints deal with updating an association or the transition of an association from one state to another. Upon updating an association, one object is disassociated from the other, and it may or may not be associated again with another object in the same class. A transition constraint can be defined in this situation to regulate how an association can be changed. Though most existing data models do not provide explicit notations or facilities for specifying a transition constraint, it is an important constraint type in database applications. We take the IDEF-1X schema S1 in Figure 4 as an example. The SSN in this schema is an alternate key of the Employee entity. Since a key is normally not updatable once its value has been given, there must exist a "non-updatable" constraint for this attribute. This non-updatable constraint is one example of transition constraints. A transition constraint can also be used to specify the relationship between the two values of an attribute before and after its update. For instance, a transition constraint can be added
to the Salary attribute of Engineer in the SDM schema S3 in Figure 4 so that an engineer's salary can only be increased and the increment should be at least 10% of its current value. A rule like this is an application dependent transition constraint and has to be defined or implemented in an application program. To represent and enforce a transition constraint in a general manner, no matter how it is actually defined or implemented and no matter what special language is used, we define the following TS macro and micro-rules.

Macro: \[ \text{TS} \left( X, \alpha, Y, \text{texp}(\alpha, \alpha_{\text{old}}) \right) \] (M6.1)

Micro:

rule \textbf{TS-01} is /* defined in class X */
triggered before this.DISASSOCIATE(\alpha, y)
action this.\alpha_{\text{old}} := y;

rule \textbf{TS-02} is /* defined in class X */
triggered before this.ASSOCIATE(\alpha, y)
condition this.\alpha_{\text{old}} \neq \text{nil} \mid \text{texp}(y, this.\alpha_{\text{old}})
otherwise REJECT;

The TS macro in M6.1 is an inter-class constraint type, which specifies a transition rule in the parameter 'texp' for the two associated classes, X and Y. These two classes are associated with each other through an association named \( \alpha \), and, as specified in the order given in the macro, \( \alpha \) serves as an attribute of X with its domain Y. The two arguments of the parameter texp (i.e., \( \alpha \) and \( \alpha_{\text{old}} \)) hold separately the current value of the attribute and the old value before the current one is assigned. These two values should satisfy the transition rule specified by texp. Otherwise, the last update of this attribute that changes its value from the value of \( \alpha_{\text{old}} \) to that of \( \alpha \) would have violated the constraint. Here we assume that the \( \alpha_{\text{old}} \) is a system-generated attribute which records the old value of attribute \( \alpha \) during an update. Using M6.1, the above example of non-updatable SSN attribute of the Employee entity can be represented as "TS(Employee, SSN, Integer, Employee.SSN = Employee.SSN_{old})". And, the example of the increase-only Salary attribute of the Engineer class can be represented by the macro "TS(Engineer, Salary, Real, Engineer.Salary \geq 1.1 * Engineer.Salary_{old})".

An attribute update is carried out in ORECOM by two primitive operations, i.e., a DISASSOCIATE operation for removing the old value and an optional ASSOCIATE operation for assigning a new value. Therefore, the enforcement of a transition rule can be achieved in two steps. First, the current attribute value has to be recorded before it is removed, and then, the rule is evaluated using the recorded value and the new value. These two steps are carried out by the micro-rules TS-01 and TS-02, respectively. Both of them are defined in the X class, i.e., the owner of the attribute \( \alpha \). The rule TS-01 simply puts the current attribute value identified by 'y'
into $\alpha_{\text{old}}$ before the DISASSOCIATE operation removes it from the X object identified by 'this'. Then, in rule TS-02, before an X object is associated with another Y object (identified by 'y') as its new attribute value, the Y object and the one recorded in $\alpha_{\text{old}}$ are evaluated together to see if this attribute update satisfies the transition rule specified in texp. If the evaluation of texp failed, the ASSOCIATE operation is rejected.

There are two things to be noted about the TS macro and its micro-rules. First, in defining the TS macro and its micro-rules, we have made an assumption that the represented transition rule applies to only those X objects whose $\alpha$ values have been assigned. For examples, the constraint of non-updatable SSN would not be applicable to employees whose SSNs have never been filled, and similarly, the constraint of at least 10% increase on Salary would not be applicable to engineers whose salaries have not been decided. In general, if an X object whose $\alpha$ attribute has never been assigned a value, then its $\alpha_{\text{old}}$ value is kept as null by default. Therefore, according to this assumption, the evaluation of the transition rule texp would not be executed unless "this.$\alpha_{\text{old}}$ ≠ nil". Second, although a TS macro contains two micro-rules, it is not necessary that both are fired for a same update. It is possible that only TS-01 is fired if an update removes an attribute value without assigning a new value. It is also possible that only TS-02 is fired if an update assigns a value to an attribute whose current value is null. For examples, an employee's SSN can be removed in one update and re-assigned in another update. The original non-updatable rule still holds even the two micro-rules, TS-01 and TS-02, are triggered by two separate updates.

The TRANSITION constraint type defined in M6.1 can be further generalized, just like the other constraint types we have discussed, so that it can be applied only to some selected subsets of objects.

3.2.6 MATHEMATICAL-DEPENDENCE (MD)

It is an inter-association constraint type. A set of attributes of a class are mathematically dependent if one attribute can be derived from the others by using a mathmatical formula. A mathematic formula can be specified with arithmetic functions and operators such as 'sin', 'log', 'sqr' and '+', '-', '*', '**', etc. It is possible for a mathematic formula to be specified in many different but mathematically equivalent ways. For example, a formula for the attributes a, b, c, and d of some class could be in the form of "a = b + c - d", or "a - b = c - d", or "a + d = b + c", and so on. Such a formula specifies a "value relationship" constraint for the related associations which needs to be maintained during data entry and update. An example of this type of constraints can be found in the SDM schema S3 shown in Figure 4. In this schema, the formula "Salary = Position * 325.3 + 20000" derives a Salary value from a Position value and it also imposes a mathematical dependence constraint upon the two attributes so that if one is updated, the other one
must also be updated accordingly. Similar formulas can also be defined in other models such as EXPRESS, OSAM*, and NIAM.

The following MD macro and the micro-rule MD-01 provide a general form to represent and enforce a mathematical dependence constraint on the associations or attributes \( \alpha_1, \alpha_2, ..., \alpha_n \) of the class \( X \). The constraint is specified as a function \( \text{mexp}(\alpha_1, ..., \alpha_n) \) which returns TRUE or FALSE. The classes \( Y_1, Y_2, ..., \) and \( Y_n \) are domains of \( \alpha_1, \alpha_2, ..., \) and \( \alpha_n \), respectively.

Macro: \( \text{MD} \ (X, ((\alpha_1, Y_1) ... (\alpha_n, Y_n)), \text{mexp}(\alpha_1, ..., \alpha_n)) \) (M7.1)

Micro:

rule MD-01 is /* defined in class X */
triggered immediate_after this.ASSOCIATE(\( \alpha_1, y_1 \), ....,
immediate_after this.ASSOCIATE(\( \alpha_n, y_n \))
condition exist this in (this AND(*\( \alpha_1 \) y_1:Y_1, .., *\( \alpha_n \) y_n:Y_n)) | mexp(y_1...y_n)
otherwise REJECT;

The value relationship between the attributes Salary and Position has the MD macro representation "MD(Engineer, ((Salary, Real), (Position, Integer)), Salary = Position * 325.3 + 20000)". This value relationship needs to be maintained whenever an engineer has both Salary and Position values. In the micro-rule MD-01, the condition clause specifies a guarded expression which is verified after an X object (i.e., this) is associated with some \( y_i \) through the association named \( \alpha_i \). The function \( \text{mexp} \) is evaluated if the X object is associated with objects \( y_1, y_2, ..., \) and \( y_n \) through the associations \( \alpha_1, \alpha_2, ..., \) and \( \alpha_n \), respectively. In the evaluation, the function \( \text{mexp}(\alpha_1, ..., \alpha_n) \) is instantiated to \( \text{mexp}(y_1, ..., y_n) \). A false result will cause the triggering ASSOCIATE operation to be rejected and aborted. We note that the trigger of the rule contains associate operations only. Disassociate and retrieve operations are not included because their execution will not affect the value relationship of the attributes.

The next representation M7.2 is a generalized MD macro, which allows a mathematical dependence constraint to be applied on subsets of objects selected by the expressions \( PY_1, PY_2, ..., \) and \( PY_n \).

Macro: \( \text{MD} \ (X, P_X, ((\alpha_1, Y_1, P_Y_1) ... (\alpha_n, Y_n, P_Y_n)), \text{mexp}(\alpha_1, ..., \alpha_n)) \) (M7.2)

3.2.7 LOGICAL-DEPENDENCE (LD)

This type of constraints specify some logical relationships among a set of associations of a class. A logical relationship can be specified in a general way using a quantified association pattern such as "forall x in x:X suchthat exist y_1, y_2 in (x OR(*\( \alpha_1 \) y_1:Y_1, *\( \alpha_2 \) y_2:Y_2))", which means every X object must have at least one of the two associations, \( \alpha_1 \) or \( \alpha_2 \), with \( Y_1 \) or \( Y_2 \) object.
example of logical relationship for the two associations between the Student class and the Full_Time_Student, Part_Time_Student classes is illustrated in the NIAM schema (S5) in Figure 4. These two associations are logically dependent on each other because of the "EXCLUSION" constraint denoted by the symbol "X". Due to this EXCLUSION constraint, the two associations can not co-existed. That is, a student can be either a full time student or a part time student, but not both. In OSAM* model, a different symbol "SX" meaning "SET-EXCLUSION" is used, and in EXPRESS, it is represented by "ENTITY Student SUPERTYPE OF (ONEOF (Full_Time_Student, Part_Time_Student));", where the key word "ONEOF" means that a student can only be in one of the subclasses. This constraint can be uniformly specified in ORECOM by the following quantified expression:

\[
\text{forall } s \text{ in (s:Student } \rightarrow \text{ Full_Time_Student)} \\
\text{suchthat } \text{ NOT exist } p \text{ in (s } \rightarrow \text{p:Part_Time_Student)}
\]

AND

\[
\text{forall } s \text{ in (s:Student } \rightarrow \text{ Part_Time_Student)} \\
\text{suchthat } \text{ NOT exist } f \text{ in (s } \rightarrow \text{f:Full_Time_Student)}
\]

(lexp1)

The NIAM schema shown in Figure 7 demonstrates another example of LOGICAL-DEPENDENCE constraints. In this schema, there is a "SUBSET" (S) constraint between the two associations Pays and Enrolls of the Student class. This constraint requires that the set of students who have paid the tuition fee must be a subset of those who have enrolled in some course(s). The subset constraint is not available in EXPRESS and IDEF-1X models, but it is equivalent to the SET-SUBSET (SS) constraint of OSAM* except that the latter is defined on two subclass associations. In the language K, it can be represented more generally as follows:

\[
\text{forall } s \text{ in (s:Student } \rightarrow \text{Pays TuitionFee)} \\
\text{suchthat } \text{ exist } c \text{ in (s } \rightarrow \text{Enrolls c:Course)}
\]

(lexp2)

The EXCLUSION and SUBSET constraints in the last two examples are both model-supported constraints. There are many user-defined constraints which are embedded in application programs can also be specified by logic expressions. For example, the following conjunctive
expression is used to represent a special constraint for the two attributes Position and Salary of the SDM schema (S3) in Figure 4. This constraint has to be implemented in a program to meet a particular application need since it can not be captured explicitly in the referenced SDM schema.

\[
\text{forall } e \text{ in (e:Engineer } \rightarrow \text{Position Integer)}
\]
\[
\quad \text{suchthat exist } r \text{ in (e } \rightarrow \text{Salary r:Real)}
\]\n\[
\text{AND}
\]
\[
\text{forall } e \text{ in (e:Engineer } \rightarrow \text{Salary Real)}
\]
\[
\quad \text{suchthat exist } i \text{ in (e } \rightarrow \text{Position i:Integer)}
\]

(lexp3)

According to the original schema, both Position and Salary are optional attributes. However, with the above constraint, if an engineer has one of these two attribute values, the other attribute must also be assigned. A similar kind of constraint, called SET-EQUALITY (SE) and EQUALITY (E), is supported by OSAM* and NIAM, respectively.

A logical dependence constraint can be used to specify the constraint on the domain of an attribute. An example of this is the first local rule defined in the WHERE clause of the EXPRESS schema S2 in Figure 4. According to this rule, a company's Fax number should always be an integer less than 9999999999. In the form of a logical expression, this can be specified as follows:

\[
\text{forall } c \text{ in (c:Company } \rightarrow \text{Fax i:INTEGER)}
\]
\[
\quad \text{suchthat (i < 9999999999)}
\]

(lexp4)

In the following, we shall introduce a more general form in terms of macro and micro-rules to represent the various logical dependence constraints on the associations α1, α2, ..., and αn.

Macro: \[\text{LD}(X,((\alpha_1,Y_1),...,((\alpha_n,Y_n))), \text{ lexp}(X,((\alpha_1,Y_1),...,((\alpha_n,Y_n)))) \] (M8.1)

Micro:

rule \textbf{LD-01} is /* defined in class X */

triggered immediate_after this.ASSOCIATE(\alpha_1, y_1), ....,

immediate_after this.ASSOCIATE(\alpha_n, y_n),

immediate_after this.DISASSOCIATE(\alpha_1, y_1), ....,

immediate_after this.DISASSOCIATE(\alpha_n, y_n)

condition \text{lexp(this, ((\alpha_1,Y_1), ..., (\alpha_n, Y_n)))}

otherwise REJECT;

The function lexp(X, α1, ..., αn) of the LD macro specifies a logical expression in the language K for associations α1, α2, ..., αn of the class X. The logic expression can be a simple or compound quantified association pattern with Boolean operators NOT, AND (Λ), OR (∨), and logic implication (⇒). Using M8.1, the above examples of logical dependence constraints can be represented as follows:
LD(Student, ((-, Full_Time_Student), (-, Part_Time_Student)), lexp1)
LD(Student, ((Pays, TuitionFee), (Enrolls, Course)), lexp2)
LD(Engineer, ((Position, Integer), (Salary, Real)), lexp3)
LD(Company, (Fax, INTEGER), lexp4)

The micro-rule LD-01 defined in the class X is the only rule for the LD macro. Its main task is to evaluate the logic expression specified in a LD macro. Since the logical relationship among the associations can be affected by any update of the associations, the micro-rule can be triggered by an ASSOCIATE or a DISASSOCIATE operation on any of the associations $\alpha_1, \alpha_2, ..., \alpha_n$. When LD-01 is triggered, the argument 'X' in the specified function $\text{lexp}(X, ((\alpha_1,Y_1), ..., (\alpha_n, Y_n)))$ is instantiated to the X object of the triggering operation, i.e., $\text{lexp}(\text{this}, ((\alpha_1,Y_1), ..., (\alpha_n, Y_n)))$. The binding of other variables for Classes $Y_1, Y_2, ..., Y_n$ still depends on their original quantifiers in the expression. The evaluation of $\text{lexp}(\text{this}, ((\alpha_1,Y_1), ..., (\alpha_n, Y_n)))$ decides if the triggering operation has to be rejected. For example, in the second LD macro shown above, if a student has enrolled in only one course and he/she has paid the tuition fee, then a disassociation of the student from his/her associated course will be rejected because it violates the "SUBSET" relationship described in the logical expression.

The macro defined in M8.1 can be extended to include some selection conditions for the involved classes as below:

Macro: \[ LD \left( X, P_X, ((\alpha_1,Y_1,P_{Y_1}), ..., (\alpha_n,Y_n,P_{Y_n})), \right. \]
\[ \left. \text{lexp}(X, (\alpha_1,Y_1), ..., (\alpha_n, Y_n)) \right) \]  \hspace{1cm} (M8.2)

4. Applications of ORECOM

In our study, we have examined a number of data models with a special emphasis on the semantics-rich models such as IDEF-1X, NIAM, EXPRESS and OSAM*. Our objective is to use the ORECOM's macro representation as a neutral representation to capture the underlying semantic properties of their modeling constructs and constraints. We have manually translated all the constructs and constraints of these models into parameterized macros each of which can be further expressed by a set of micro-rules representing the operational semantics of a DBMS. Additionally, we have implemented a schema translation system to demonstrate the workability of schema translation through this neutral representation. In this section, we shall use some selected constructs and constraints of the above mentioned four data models as examples to illustrate the concept of semantic decomposition and the technique of schema translation which is described in detail in a separate paper [SU92]. Detailed decompositions of these models are available in [FANG93]. Some other potential applications of ORECOM will also be discussed.
4.1 Analysis of Data Models

4.1.1 IDEF-1X

IDEF-1X [LOOM86, 87] is an extension of the data model IDEF-1 (or Integrated Computer-Aided Manufacturing Definition Method 1). IDEF-1 was developed in the late 1970's under the auspices of the U.S. Air Force, and later became one of the best known data modeling techniques in the industry. This model is a hybrid of the ER model and the relational model. It uses the concepts of entities, attributes, entity relationships to express data semantics and provides a nice graphical notation for representing some structural properties and constraints. Figure 8 shows some examples of IDEF-1X construct and constraint patterns and their corresponding macro representations. Construct patterns I-C-01 to I-T-04-2 describe an entity, an attribute, and an alternate-key attribute, respectively. These concepts are commonly supported in other data models even though different terminologies and notations have been used. For example, corresponding to an IDEF-1X entity, NIAM uses non-lexical object type (NOLOT) and OSAM* uses entity class (E-class). An IDEF-1X alternate-key is captured in EXPRESS and NIAM by a uniqueness constraint.

The construct I-C-01 of Figure 8 is an IDEF-1X entity whose semantic properties can be represented in ORECOM by a Membership (MB) macro. This macro states that the IDEF-1X entity, X, can be mapped to an entity class of ORECOM, whose members are system-named objects and has an external identifier (Y1, ..., Yn), i.e., the composite primary key of X. The default object structure of X is 'simple' (denoted by the first '-' in the macro), and the class type is
'E-CLASS' because it defines a set of system-named objects. It does not have a membership constraint nor method specifications (denoted by the last two '-'s in the macro). The pattern I-C-05 captures the constraints of an **attribute** (Y), which relates instances of an entity class (X) to instances of the underlying domain Y (i.e., dom(Y)). The symbol "(O)" after Y means that it is an optional attribute which is represented in ORECOM by a partial participation constraint: the first Participation (PT) macro of I-C-05. On the other hand, not all instances of the domain of Y are associated with X instances, which is also a partial participation constraint represented by the second Participation (PT) macro. The mapping between entity class X and the domain class of Y is many-to-one which is captured by the Cardinality (CD) macro of I-C-05. If Y is an **alternate key** (AK) of X as shown in I-T-04-2, then the macros in I-C-05 need to be modified to capture the "non-null" (or total participation) constraint and the "uniqueness" (or one-to-one cardinality mapping) constraint associated with an alternate-key attribute. We note here that the concepts of a primary key and an alternate key can be similarly decomposed into participation and cardinality constraints.

Figure 8. Examples of IDEF-1X construct and constraint patterns.
The last two constructs of Figure 8, I-C-05 and I-T-06-1, are two examples of entity relationship between entities X and Y which is called in IDEF-1X the **connection relationship**. The former (see I-C-05) is graphically represented in IDEF-1X by a dashed line labelled with a verb 'R', and the primary key of entity Y is a foreign key (FK) of X. In ORECOM, we treat the inverse of R (i.e., \( R^{-1} \)) as an attribute of X and the domain of \( R^{-1} \) is Y. In I-C-05, the foreign key Z is located below the line of entity X and is an optional attribute, which means that instances of X do not have to be associated with any Y instance (i.e., no **existence dependency**). This semantic property is captured in ORECOM by the first Participation (PT) macro. The black dot on the X side of the link indicates that a Y instance can be connected to zero, one, or many instances of X, which implies that Y is partially participated in the association R with X. This property is captured by the second Participation (PT) macro. The cardinality mapping from X to Y in I-C-05 is many-to-one as captured by the Cardinality (CD) macro.

Compared with I-C-05, the construct I-T-06-1 is more constrained in two ways. First, the foreign key becomes a part of the primary key of X, which enforces an **identifier dependency** constraint and hence, an existence dependency constraint. Second, every Y instance must be associated with at least one X instance (indicated by the symbol "P"). The identifier dependency is depicted in IDEF-1X by a solid line instead of the dashed line in I-C-05, and the entity box of X is also changed to a round-cornered box which indicates that the entity X is identifier-dependent on at least one other entity (e.g., entity Y, in I-T-06-1). This constraint is represented by changing the first Participation (PT) macro of I-C-05 to a total participation constraint. The constraint imposed by the symbol "P" is represented by changing the second Participation (PT) macro of I-C-05 to a total participation. The cardinality is not changed, i.e., it is still many-to-one. In a connection relationship in IDEF-1X, a label "Z" can be used in the place of "P" to mean zero or one Y instance's connection to X instance. This constraint can be similarly expressed in a cardinality macro with different parameter values.

4.1.2 EXPRESS

EXPRESS [SCHE89, ISO92] is an information modeling language which is a strong candidate for an international standard for product specification. An EXPRESS schema defines data types and their constraints. The definition of a data type 'positive' is shown below:

```plaintext
    TYPE  positive = INTEGER;
    WHERE  self > 0;
END_TYPE;
```

The above is called a defined data type which is similar to the domain class (D-class) of OSAM* and the lexical object type (LOT) of NIAM. However, it is not explicitly supported by IDEF-1X since all domains of attributes are hidden from an IDEF-1X schema. Figure 9 shows some general constructs of EXPRESS. The X in the construct E-C-08 is a defined data type whose domain is Y
(which is implicitly a simple data type in EXPRESS). A defined data type can have a domain rule specified by an expression 'exp' in the WHERE clause. In ORECOM, this construct can be represented by a Membership (MB) macro having X as a domain class (i.e., containing self-naming objects because the domain Y is a simple type containing self-naming objects) and the expression exp as its membership constraint. Since every member of the defined type X must be an instance of Y that satisfies exp, and every such qualified Y instance becomes automatically a member of X, two Participation (PT) macros are used to describe these two total participation constraints. In addition, a Cardinality (CD) macro is used to capture the one-to-one mapping relationship between X and Y.

For defining complex data types, EXPRESS provides four "aggregations" (i.e., SET, LIST, BAG, and ARRAY), which can be used in any combination and in any length (e.g., LIST of ARRAY of ARRAY of SET of INTEGER). An equivalent feature of this is supported in OSAM* but it is not generally available in other existing models. In EXPRESS, aggregations can be used in the TYPE declaration for defined data types or in the entity declaration for defining complex domains of attributes. In either case, each aggregation of a complex data type is viewed in ORECOM as defining a new class from the base (or domain) of that aggregation, whose members are complex objects having the structure specified by the aggregation. For example, the "SET of INTEGER" in the above example will be treated as defining a new class from INTEGER, say,
SET_INTEGER, and each object of this new class represents a set of INTEGER objects. Based on this SET_INTEGER, a higher level aggregation such as "ARRAY OF SET OF INTEGER" would define another new class, ARRAY_SET_INTEGER, whose members represent arrays of objects of the class SET_INTEGER. The general construct shown in E-C-01 of Figure 9 is EXPRESS's way of specifying an aggregation of X with a minimum of zero and a maximum of u number of X. It can be translated to a number of macros. The Membership macro specifies that a new created class named Y is defined as an aggregation of X and is a domain class. (Here the X is assumed to be a simple data type or an aggregation of simple data type in EXPRESS.) The 'agg' in E-C-01 can be SET, LIST, or BAG only, because the lower and upper bounds of the other aggregation ARRAY mean differently and need to be represented by a separate construct. The lower bound of the 'agg' in E-C-01 is zero, which means that an instance of X may not contain any Y instance, and also, as a default condition, not every Y instance has to become an element of some aggregated object of X. Both of these two conditions are partial participation constraints and therefore can be represented by the two Participation macros in E-C-01. The upper bound (u) of the 'agg' determines the cardinality mapping between X and Y as many-to-u (or M-to-u) as shown in E-C-01. By changing the parameters of the macros associated with this construct, a number of other similar EXPRESS constructs such as an aggregation with a non-zero lower bound and a LIST of unique elements, can be represented.

Besides TYPE declarations, entity declarations are the major part of an EXPRESS schema. The definition of an EXPRESS entity type is in terms of its properties (or attributes), each of which has an associated domain and an optional constraint of the domain. An attribute can be further constrained to be a non-optional attribute, a unique attribute, or a derived attribute. The first two types of constraints are similar to the non-null attribute and alternate-key attribute of IDEF-1X which have been discussed previously. A derived attribute is shown in E-T-1516-3 of Figure 9, in which the value of attribute Y is derived by the expression 'exp(Z1...Zn)'. Since the domain of Y (i.e., [agg] W) can be a simple data type, a defined data type (specified in the TYPE section), an entity type, or a complex data type specified by an aggregation of W, we simply use 'dom(Y)' as the general representation of the domain of Y. To capture the value relationship between the values of Y (i.e., members of [agg]W) and the domains of attributes Zs which derive Y values, a Mathematical-Dependence (MD) macro is used to specify "Y = exp(Z1...Zn)".

Another way to specify a constraint on an attribute in EXPRESS is to define a "local rule" in a WHERE clause as illustrated in E-T-1516-5 of Figure 9. It specifies that the values of attributes Z1, Z2, ..., and Zn of each entity of X have to satisfy a local rule expressed by an expression exp(Z1...Zn). Two examples of local rules are: "Z1 - 10 > Z2 + Z3" and "Z1 :=: Z2". The symbol ":=:" in the second expression represents an instance equality operator for two entity typed attributes. This expression returns TRUE if both Z1 and Z2 refer to the same entity instance.
Similar rule specification facility is also available in OSAM*, but each has its own rule language syntax. In E-T-1516-5, the local rule is represented in ORECOM as an inter-association constraint using a Logical-Dependence (LD) macro as shown in Figure 9.

4.1.3 NIAM

NIAM is an information modeling methodology pioneered by G. M. Nijssen [VERH82]. This model is sometimes referred to as a "binary semantic model" because it provides a binary representation of data, semantics, and constraint. The building blocks of NIAM are lexical objects (LOTs), non-lexical objects (NOLOTs), associations between LOTs and NOLOTs (called BRIDGEs), and associations between different NOLOTs (called IDEAs). Furthermore, both BRIDGEs and IDEAs are composed of a pair of ROLEs which are usually verbs that describe the semantics of the associations. The most interesting feature of NIAM is that it supports many kinds of constraints on multiple associations. Besides the UNIQUENESS (U) constraint, which is similar to the alternate-key (AK) of IDEF-1X or the UNIQUE constraint of EXPRESS, there are three association constraints: EQUALITY (E), EXCLUSION (X), and SUBSET (S). These constraints are described separately in the patterns of N-C-08, N-C-10, and N-C-15 in Figure 10. In these patterns, the NOLOTs Y1 and Y2 are assumed to be the domains of attributes a1 and a2 of X, respectively. We show these constraints in a pair of IDEAs even though they can also exist in a pair of BRIDGEs or between an IDEA and a BRIDGE. For each construct, a Logical-Dependence (LD) macro is used to capture the constraint in ORECOM's representation. Since the constraints are different, the logic expressions of their corresponding macros are different. For the constraint of EQUALITY (E) in N-C-08, objects of X must either have both values of a1 and a2 or none of them. Therefore, the logical expression for N-C-08 would be as follows:

\[
\text{AND} \quad \text{(forall } x \text{ in } (x:X \rightarrow a_1 \text{ } Y_1) \text{ suchthat exist } y_2 \text{ in } (x \rightarrow a_2 \text{ } y_2:Y_2)) \\quad \text{AND} \quad \text{(forall } x \text{ in } (x:X \rightarrow a_2 \text{ } Y_2) \text{ suchthat exist } y_1 \text{ in } (x \rightarrow a_1 \text{ } y_1:Y_1))
\]

This conjunctive association pattern expression ensures that there does not exist any X object which associates with only one of the two classes, Y1 or Y2. As we discussed in the section on the LD macro, the 'X's in the above expression will be bound to the X object of a triggering operation which triggers the micro-rule LD-01 to evaluate the expression.

The constraint of EXCLUSION (X) shown in N-C-10 requires that no object of X can have both values of a1 and a2 at the same time. The logical expression for this constraint in the Macro representation is:

\[
\text{AND} \quad \text{(forall } x \text{ in } (x:X \rightarrow a_1 \text{ } Y_1) \text{ suchthat NOT exist } y_2 \text{ in } (x \rightarrow a_2 \text{ } y_2:Y_2)) \\quad \text{AND} \quad \text{(forall } x \text{ in } (x:X \rightarrow a_2 \text{ } Y_2) \text{ suchthat NOT exist } y_1 \text{ in } (x \rightarrow a_1 \text{ } y_1:Y_1))
\]
The semantics of the SUBSET (S) constraint of N-C-15 is that the set of X objects which have an a1 value is a subset of those X object which have an a2 value, or equivalently, the association a1 between an X object and a Y1 object implies the association a2 between that X object and a Y2 object. The logical expression for this constraint is shown below.

\[(\forall x \in (x:X \rightarrow^a1 Y1 \text{ suchthat exist } y2 \in (x \rightarrow^a2 y2:Y2)) \text{ AND} \]

\[(\forall x \in (x:X \rightarrow^a2 Y2 \text{ suchthat exist } y1 \in (x \rightarrow^a1 y1:Y1))\]

\[(\forall x \in (x:X \rightarrow^a1 Y1 \text{ suchthat exist } y2 \in (x \rightarrow^a2 y2:Y2))\]

\[(\forall x \in (x:X \rightarrow^a2 Y2 \text{ suchthat exist } y1 \in (x \rightarrow^a1 y1:Y1))\]
In addition to the above inter-association constraints, NIAM allows object types to form a supertype-subtype hierarchy. This important concept is supported in almost every new semantic or object-oriented data model, e.g., the generalization (G) of OSAM*, the supertype-subtype of EXPRESS and IDEF-1X. What is shown in the pattern of N-C-15 is a supertype-subtype constraint of NIAM between the NOLOTs X and Y. Since X is the supertype of Y, Y inherits all properties of X. The inheritance semantics of this construct is represented by the Inheritance (IH) macro of N-C-15. The two Participation (PT) macros capture the partial participation of X with Y and the total participation of Y with X, and the Cardinality (CD) macro captures the one-to-one mapping between X and Y.

In NIAM, a set of subtype associations may have two kinds of constraints: TOTALITY (T) and DISJOINT ($\neq$). The construct N-T-16-1 shows a TOTALITY (T) constraint on the subtypes (Y1, Y2, ..., Yn). This constraint states that the union of all the subtype objects should be equal to the set of objects of X. In other words, X is totally participated in the set of subtypes (Y1, Y2, ..., Yn). This TOTALITY (T) constraint is called a "total specialization" in OSAM* and IDEF-1X. In ORECOM, it is neutrally represented by an inter-association Participation (PT) macro as shown in N-T-16-1. The second type of constraint among subtypes is DISJOINT ($\neq$) as shown in N-T-16-2. It specifies that the objects of each subtype of X can not overlap. This constraint is similar to the EXCLUSION (X) of N-C-10 except it is applicable to subtype classes only. The DISJOINT constraint is also available in OSAM* (i.e., the set-exclusion or SX constraint) and EXPRESS (i.e., the ONEOF constraint). In IDEF-1X, however, it is defined as a default constraint among subtype entities. In ORECOM, the DISJOINT ($\neq$) constraint specifies a logical relationship among subtypes and therefore is captured by a Logical-Dependence (LD) macro containing a logic expression as specified in Figure 10.

4.1.4 OSAM*

The OSAM* [SU89] is an object-oriented semantic association model developed at the Database Systems Research and Development Center of the University of Florida. The basic structural modeling concepts of this model are object classes (i.e., E-class and D-class) and associations between/among classes. There are five system-defined association types in OSAM* to represent different object/class relationships. They are aggregation (A), generalization (G), interaction (I), composition (C), and cross-product (X) associations. The A-association is similar to the attribute of EXPRESS, the attribute and connection relationship of IDEF-1X, and the BRIDGE (connecting one LOT and one NOLOT) and the IDEA (connecting two NOLOTs) of NIAM. The G-association is identical to the supertype-subtype relation of these models except a few different optional constraints. The I-association is a special association which models the interactions among a set of entity classes and is similar to the relationship construct of the ER
model. For example, the interactions among Student, Instructor, and Course classes can be modeled as objects of another class called Registration. In the construct shown in O-C-15 of Figure 11, the entity class X is defined by an interaction among a number of classes including class Y. Z is an optional name of the association between X and Y. Three macros are needed for specifying the constraints between X and Y or X and any other constituent class. The first macro represents a total participation constraint so that every X instance is existence dependent on each of its constituent class' instances. (It is not meaningful to record an interaction among some objects if they do not exist as members of their corresponding classes.) The second macro is a partial participation because not every Y instance has to be interacted with instances of other classes. The cardinality mapping between X and Y is many-to-one (i.e., a Y instance can participate in many interactions with the object instances of other constituent classes) as captured by the Cardinality (CD) macro. If all Y instances have to participate in some interactions with other instances as defined by instances of X, then a key word "TP" is specified above the class Y (see O-T-15-1) and the second Participation macro of O-C-15 will be replaced by a total participation macro. An

<table>
<thead>
<tr>
<th>No.</th>
<th>Pattern</th>
<th>Macro Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>O-C-15</td>
<td><img src="image1" alt="Diagram" /></td>
<td>PT( X, -, count(X), count(X), (Z, Y, -) )&lt;br&gt;PT( Y, -, 0, count(Y), (Z', X, -) )&lt;br&gt;CD((-), (X, -), (Z, Y, -), 1, M, 1, 1 )</td>
</tr>
<tr>
<td>O-T-15-1</td>
<td><img src="image2" alt="Diagram" /></td>
<td>PT( X, -, count(X), count(X), (Z, Y, -) )&lt;br&gt;PT( Y, -, count(Y), count(Y), (Z', X, -) )&lt;br&gt;CD((-), (X, -), (Z, Y, -), 1, M, 1, 1 )</td>
</tr>
<tr>
<td>O-T-15-2</td>
<td><img src="image3" alt="Diagram" /></td>
<td>CD((Z1, Y1, -), (X, -), (Z2, Y2, -), p1, q1, p2, q2 )</td>
</tr>
<tr>
<td>O-C-16</td>
<td><img src="image4" alt="Diagram" /></td>
<td>MB( X, system-named, -, SET, ENTITY_OBJECT, Px, - ) where Px = (exist x in (x:X where x = Instance(Y1))) AND .......... AND (exist x in (x:X where x = Instance(Yn)))</td>
</tr>
</tbody>
</table>

Figure 11. Examples of OSAM* construct and constraint patterns.
important characteristic of the I-association is that an indirect cardinality mapping constraint can be added for a pair of interacting classes. As shown in O-T-15-2, the indirect mapping between $Y_1$ and $Y_2$ is $[p_1, q_1]$-to-$[p_2, q_2]$, that is, every $Y_1$ instance can participate in interactions with at least $p_2$ and at most $q_2$ of $Y_2$ instances, and every $Y_2$ instance can participate in interactions with at least $p_1$ and at most $q_1$ of $Y_1$ instances. To represent this indirect mapping constraint in ORECOM, an inter-association Cardinality (CD) macro is used. Note that, in this construct, the constraints of each individual pair ($X$ and one of its constituent classes) is supposed to be captured by constructs of O-C-15 or O-T-15-1.

The last example OSAM* construct in Figure 11 (i.e., O-C-16) is particularly useful for statistical database applications. This construct represents a Composition or C-association. It means that the dynamic sets of objects in classes $Y_1, Y_2, ..., Y_n$ are instances of $X$. Any attribute (usually statistical summary attribute) of class $X$ (defined by an aggregation association not shown in O-C-16) would characterize the set-structured instances rather than the individual members in the sets. As a consequence, the Membership (MB) macro representation of this construct shows that $X$ is a system-named class, its structure is SET and its class type is ENTITY_OBJECT where ENTITY_OBJECT is a system-defined class of all entity objects of a database. The constraint on the members of $X$ is that each member of $X$ corresponds to the entire set of instances of $Y_i$ for $i = 1..n$. Here, Instance($Y_i$) denotes the entire set of instances of $Y_i$.

### 4.2 Applications

The neutral data model ORECOM together with the presented technique of semantic decomposition offer a general framework for resolving the data model heterogeneity problem found in multimodel database systems. Many problems associated with the interoperability of multimodel database systems such as data model learning, schema translation, schema integration, and schema verification and optimization can all be benefited from it.

#### 4.2.1 Data Model Learning

One possible application of ORECOM is to assist a user or a database system designer or developer in learning the semantics of a new data model in two ways. First, when a data model is mapped to an ORECOM representation, each of its modeling constructs and its associated constraints are fully decomposed (as presented in the previous sections) into a concise parameterized representation (i.e., the macro representation) and its corresponding operational semantics (i.e., the micro-rule representation). A user can learn quite easily the modeling concepts of a new data model from the macro representations of its constructs and a system designer or developer can gain precise information about the DBMS operations needed to implement a construct or enforce a constraint from its micro-rule representation. This information is useful in designing a schema, implementing a database, or writing a control or interface program. Second,
since different data models can be mapped to the neutral ORECOM representation, the data model learning process can be considerably eased by comparing the neutral representation of a new construct or constraint with that of the corresponding construct or constraint in a model familiar to the user. Due to the neutral representation of all decomposed modeling constructs and constraints, a cross reference between a new model and a well-understood model can be automatically generated to compare their commonalities and differences.

4.2.2 Schema Translation

Schema translation is an important and necessary process to achieve data sharing among different databases in a heterogeneous database system. It is a prerequisite for database translation, view conversion, query translation, and global transaction management. A major issue in schema translation is whether the target schema preserves the original semantics of the source schema in a translation. The primitive semantic representations of ORECOM facilitate a semantics preserving schema translation in the following way. By decomposing the modeling constructs and constraints specified in the source and target schemata into the common and primitive ORECOM representations (as described in Section 4.1), the equivalences and differences between their modeling constructs or constraints can be determined precisely by matching their ORECOM representations. The derived equivalence relationship is called an equivalence matrix of the source and target data models. Shown in Figure 12 is a tabular form of an equivalence matrix for translating EXPRESS to IDEF-1X. Discrepancies found between the source and target schemata can be explicitly specified by the unmatched primitive representations (either in terms of macros or micro-rules) and be used in the application development to account for the missing semantics. For example, if the pattern E-C-13, which represents an EXPRESS entity type, appears in a source schema, it will be translated into I-C-01, which defines an IDEF-1X entity. This particular translation needs an adjustment due to their different external identifiers requirements, which is explicitly specified in terms of the two MEMBERSHIP (MB) macros. The symbol "—" in the table means there is no corresponding target construct or constraint to a source item. A schema translation system can be guided by a set of equivalence matrices each of which defines the mapping between the ORECOM representations of one data model and those of another model. This approach has been successfully used in the development of a data model and schema translation system at the Database Systems Research and Development Center of the University of Florida [SU92]. The translation system is capable of translating schemata defined in EXPRESS, IDEF-1X, NIAM, and OSAM*. It has been demonstrated at the EXPRESS User's Group meeting in Dallas, Texas, October 17-18, 1992 and to many industrial companies. Readers who are interested in the system can contact the first author of this paper.
4.2.3 Schema Integration

In a multimodel database system environment, schema integration is a necessary task in the development of a federated or integrated database management system. A desirable approach to schema integration is to first translate heterogeneous schemata into common representations and then integrate them. This approach is used in the work reported in [SPAC92]. The use of ORECOM as the neutral model has the following benefits: 1) the integration can be carried out on the basis of macros whose low-level and primitive representations can distinguish the fine semantic differences of different modeling constructs and constraints which can not be distinguished by a high-level common model, 2) a tightly integrated global schema can be generated to capture not only the structures but also the constraints and operations of the component schemata, 3) the mapping relationship between the global and the component schemata can be recorded and reported.
in detail (in terms of macros or micro-rules) for interfacing the global and local query and transaction processes.

4.2.4 Schema Verification and Optimization

The existing tools for checking the correctness of a schema (i.e., verification) and for removing the redundant constructs or constraints from a schema (i.e., optimization) are usually data-model-dependent. Using ORECOM as a neutral model, the development of a common schema management tool for these two tasks is possible because heterogeneous schemata can be translated into the primitive ORECOM representations before applying verification and optimization techniques on them.

5. Conclusions

Heterogeneous, multimodel databases have become and will continue to be an important area of database research in supporting non-traditional database applications such as office automation, integrated manufacturing, military command/control/communication, multimedia data management, scientific databases, etc. In order to achieve the interoperability of heterogeneous database systems, a semantics-preserving data model and schema translation is a necessity. Our development of such a translation system is based upon two basic principles. First, to reduce the complexity and to avoid a large number of pair-wise direct translations, a neutral data model is used as the intermediate representation in data model and schema translations. Second, to deal with the syntactic and semantic heterogeneity problem, high-level modeling constructs and constraints are decomposed into some low-level, neutral, primitive semantic representations so that the equivalence relationship between different constructs and constraints can be determined precisely and specified explicitly.

In this paper, we have presented a low-level model ORECOM as the neutral representation based on our analysis of several semantically rich data models. The key features of this model are its general structural constructs for representing the structural properties of all things of interest to a database application in terms of objects, classes, and associations, and its powerful behavioral constructs specified in terms of object operations and micro-rules. We also defined formally eight basic constraint types and their variations found in many existing data models using parameterized macros and their corresponding micro-rules. These basic constraint types have been used as the neutral representations of the high-level modeling constructs and constraints of several semantic-rich data models. Their utility in schema translation has been demonstrated in an implemented data model and schema translation system developed at the University of Florida.

It is the authors' hope that this work will not only contribute to our understanding of the semantics of data model constructs and constraints but also provide a solid foundation for solving
many problems related to the interoperability of multimodel multidatabase systems such as data
model learning, schema translation, schema integration, and schema verification and optimization.

REFERENCES

[ACM91] ACM SIGMOD RECORD, special issue on the semantics of multidatabase systems, vol. 20, no. 4,
1989 ACM SIGMOD Int'l Conf. on Management of Data, 1989.
programming language," Masters thesis, Electrical Engineering Department, University of Florida,
[BREI90] Y. Breitbart, "Multidatabase Interoperability," ACM SIGMOD RECORD, vol. 19, no. 3, September,
1990.
[CHEN76] P. P. Chen, "The Entity-Relationship model - toward a unified view of data," ACM Transactions on
[ELMA90] A. K. Elmagarmid, C. Pu, "Guest Editors' Introduction to the Special Issue on Heterogeneous
and Schema Translation System," Ph. D. dissertation, Electrical Engineering Department, University
[GANG91] D. Gangopadhyay, T. Barsalou, "On the Semantic Equivalence of Heterogeneous Representations in
[HAMM81] M. Hammer, D. Mcleod, "Database description with SDM: A semantic database model," ACM TODS,
vol. 6, no. 3, 1981.
Computer and Information Science Department, University of Florida, June, 1992, (submitted to The
Vldb Journal, June, 1992)
[LOHM91] G. M. Lohman, B. Lindsay, H. Pirahesh, K. B. Schiefer, "Extensions to Starburst: objects, types,