Schema Integration Methodologies for Multidatabases and the Relational Integration Model - Candidacy document

Ramon Lawrence
Department of Computer Science
University of Manitoba
umlawren@cs.umanitoba.ca

March 22, 1999

Abstract

Although considerable effort has been placed on integrating schemas of diverse databases, the problem remains largely unsolved due to its complexity. Schema integration is hard because at some level, both the operational and data semantics of a database need to be known for integration to be successful. This work studies the schema integration problem as applied to multidatabases. The major contribution of the work is a survey of proposed models and methods that capture data semantics and simplify schema integration, and a discussion on why they are insufficient. By examining previous work, we define a set of desirable properties that an integration methodology should have. Finally, we propose a new language for capturing data semantics and integrating simple relational schemas.

1 Introduction

Data collection and integration are rapidly becoming the most important issues in the information systems area. The basic problem is insuring that the vast amounts of data collected by various legacy systems can be used in a meaningful and integrated way. Currently, most organizations use only a small fraction of the data gathered by their systems for a variety of reasons. These reasons include the difficulty of getting older systems to interoperate with each other, and the complexity of combining many different data sources into a coherent whole.

A multidatabase system (MDBS) is a collection of databases interconnected to share data. One of the major tasks in building a multidatabase is determining and integrating the data from the component database systems into a coherent global view. Automating the extraction and integration of this data is difficult because the semantics of the data are not fully captured by

its organization and data schema. The schema integration problem is the problem associated with combining the diverse schemas of the different databases into a coherent global view by reconciling any structural or semantic conflicts between the component databases.

Integrating diverse data sources is becoming increasingly important. Schema integration is involved in constructing both a multidatabase system (MDBS) and a data warehouse (DW). Both of these architectures are finding more applications in industry because they allow high-level personnel transparent access to data across multiple sites and provide a uniform, encompassing view of the data in an organization. A uniform view allows managers to examine historical information and solve problems at a higher level. Thus, a method for simplifying schema integration would be of great theoretical and practical importance.

The literature has proposed various methods for capturing data semantics. This work will examine some of these methods and their characteristics. The variety of approaches to solve the problem range from meta-model definition, schema re-engineering, using lexical semantics, or artificial intelligence techniques. Industrial-level work is based on defining languages and protocols to aid the integration process.

Previous work has resulted in limited success. Many methods require user input to specify missing semantic information or rely on heuristic rules which cannot be automated. The focus of this work is to study the previous work and determine where the shortcomings lie. Using these results, we define a set of desirable features which an integration method must have. Finally, we propose an integration language which integrates relational schemas.

In this paper, we have defined an integration taxonomy which characterizes schema integration methodologies according to the properties of: resolved conflicts, transparency, and automation level. The ability of an integration methodology to resolve conflicts is fundamental as different schemas will be developed under different data models and ideologies. This results

in both structural and semantic conflicts between schemas which must be resolved to construct a global view. Transparency is important as the integration should be performed transparently to the user and the application programmer. Forcing the user or programmer to be aware of the integration or even perform the integration, makes applications susceptible to change and increases complexity. Finally, integration techniques should be automatic in nature requiring minimal user interaction. User-level integration is tedious and slow, and an integration methodology should try to hide the integration complexity from the users and designers as much as possible.

Various integration methods satisfy some of the taxonomy properties. Many of the early semantic modeling techniques are very good at resolving all types of semantic conflicts. Unfortunately, these techniques are based on heuristic algorithms with little transparency or automation. More recent techniques sacrifice transparency by forcing the programmer or designer to dynamically resolve conflicts using MDBS query languages. These techniques represent a workable solution in industrial environments but suffer from poor transparency and automation.

This work proposes the Relational Integration Model (RIM) for schema integration. RIM is designed to integrate simple relational schemas which meet certain criteria. By automatically extracting and storing metadata at the relation-level, RIM is able to resolve integration conflicts automatically with no user intervention. The product of the integration is a global view, and the conflict resolution is performed without user involvement resulting in a highly transparent and automatic system.

Key features of RIM include:

- Performs dynamic integration as schemas are integrated one at a time into the global view and as needed.
- Captures both operational and structural metadata and uses it to resolve conflicts.
- Performs automatic conflict resolution.

- Metadata is captured only at the local level and some is extracted automatically, which removes the designer from most integration-related problems.
- Local schemas are merged into a global view using a global dictionary to resolve naming conflicts.

The project timeline is as follows:

- Take candidacy exam in March 1999.
- By August 1999, main work on RIM will be completed.
- By December 1999, thesis will be written and defended.

If work on RIM should prove unsuccessful, the integration work can be used as a basis for defining and categorizing the problems with integration in terms of trust between information sources.

2 Background work

2.1 Multidatabase Architecture

A **multidatabase** is a collection of autonomous databases participating in a global federation that are capable of exchanging data. The basic multidatabase system (MDBS) architecture (see Figure 1) consists of a global transaction manager (GTM) which handles the execution of global transactions (GTs) and is responsible for dividing them into subtransactions (STs) for submission to the local database systems (LDBSs) participating in the MDBS. Each LDBS in the system has an associated global transaction server (GTS) which converts the subtransactions issued by the GTM into a form usable by the LDBS. Certain algorithms also rely on the GTS for concurrency control and simulating features, such as visible 2PC, that the LDBS does not provide. Each LDBS is autonomous so modifications are not allowed. Many proposed algorithms assume the LDBS exhibits certain properties which may violate the principle of full

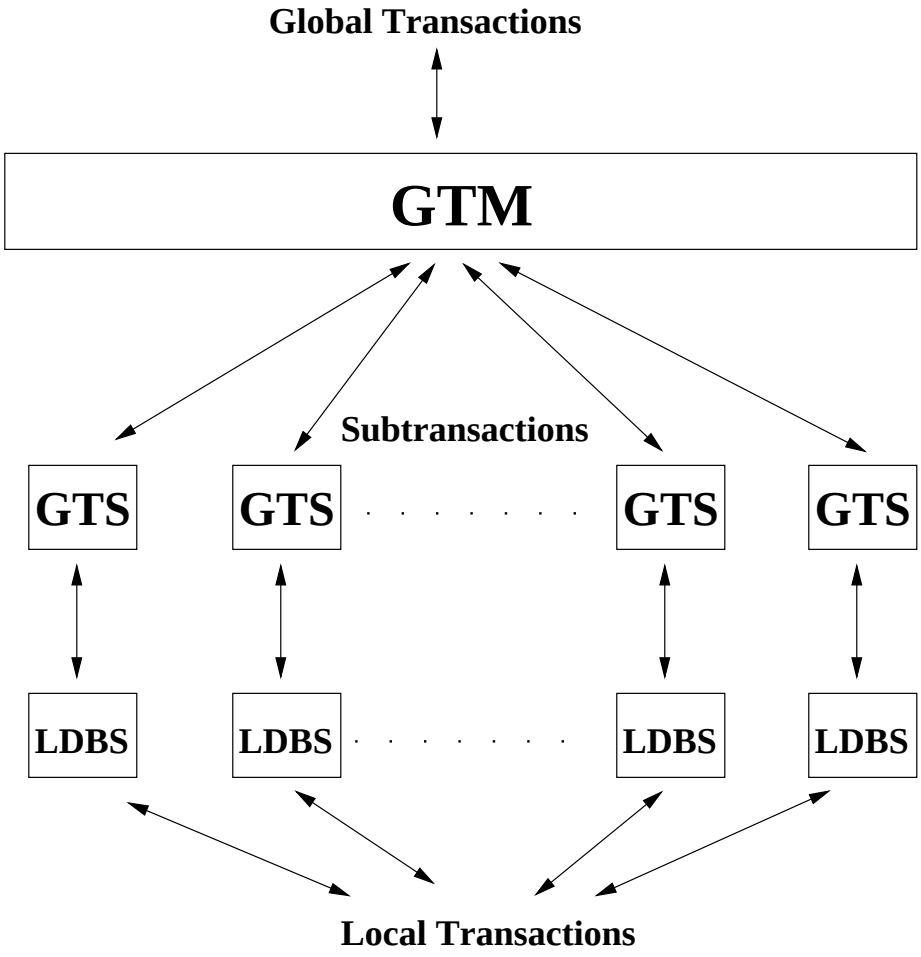

Figure 1: MDBS Architecture

autonomy. Finally, local transactions, not under the control of the GTM, are allowed at each LDBS as if the LDBS was not participating in the MDBS.

Global transactions are posed to the global view which is constructed by integrating the local views provided by each LDBS. The local view provided to the global view may be a subset of the entire local view for the LDBS. Schema integration involves taking the local views provided by the LDBSs and combining them into a global view by resolving any conflicts between the views.

Good surveys of the issues involved in the construction and operation of multidatabase systems and federated systems appear in [29] and [22].

2.2 The Integration Problem

The integration problem refers to the problem associated with integrating the data from two or more different data sources. Integration is often required between applications or databases with widely differing views on the data and how it is organized. Thus, integration is hard because conflicts at both the structural and semantic level must be addressed. Further complicating the problem is that most systems do not explicitly capture semantic information. This forces designers performing the integration to impose assumptions on the data and manually integrate various data sources based on those assumptions. The integration problem is related to the view integration problem in databases [15].

The two basic levels of integration are schema-level integration and data-level integration. The higher-level of integration is at the schema level. The schema contains the format, structure, and organization of the data in a system. Most systems do not capture operational or data semantics at the schema level. Schema integration is the process of combining database schemas into a coherent, global view. After schema integration has been completed, there is the related problem of data integration. Data integration focuses on integrating information at the data item level. This involves such tasks as comparing keys to insure that they represent the same entity in different databases, handling spatial or temporal data, or combining similar data items stored in different data formats. This work focuses on the schema integration problem.

2.2.1 The Schema Integration Problem

Schema integration is the process of combining database schemas into a coherent, global view. Since database systems tend to be developed in isolation, it is often hard to combine different database schemas because of different data models or structural differences in how the data is represented and stored. It is difficult to determine when two schemas represent the same

data in different databases, even if they were developed under the same data model. Each database system has its own world view, and the task of integrating these "views" is as difficult as integrating the knowledge of two humans. Thus, there are many factors which may cause schema diversity[1]:

- Different user or view perspectives
- Equivalence among constructs of the model (eg. a designer uses an attribute in one schema and an entity in another)
- Incompatible design specifications (eg. cardinality constraints, bad name choices)
- Common concepts can be represented by different representations which may be identical, equivalent, compatible, or incompatible
- Concepts may be related by some semantic property which arises only by combining the schemas (interschema properties)

There are several features of schema integration which make it difficult. The first problem is that data may be represented using different data models. For example, one database may use the relational model, while another database may use an object-oriented model. Even when two databases use the same data model, both naming and structural conflicts may arise.

Naming conflicts occur when the same data is stored in multiple databases, but it is referred to by different names. Naming conflicts arise when names are homonyms and when names are synonyms. The homonym naming problem is when the same name is used for two different concepts. The synonym naming problem occurs when the same concept is described using two or more different names.

Structural conflicts arise when data is organized using different model constructs or integrity constraints. Some common structural conflicts[1] are:

- **Type conflicts** - using different model constructs to represent the same data
- **Dependency conflicts** - group of concepts related differently in different schemas (eg. 1-to-1 participation versus 1-to-N participation)

- **Key conflicts** - different keys for the same entity
- **Behavioral conflicts** - different insertion/deletion policies for the same entity
- **Interschema properties** - schema properties that only arise when two or more schemas are combined

A robust integration methodology must be able to handle both naming and structural conflicts. Subsequent sections describe many proposed methods to solve the schema integration problem.

2.2.2 The Data Integration Problem

Data integration is the process of combining data at the entity-level. After schema integration has been completed, a uniform global view has been constructed. However, it may be difficult to combine all the data instances in the combined schemas in a meaningful way. Combining the data instances is the focus of data integration.

Data integration is difficult because similar data entities in different databases may not have the same key. For example, an employee may be uniquely identified by name in one database, and by social insurance number in another. Determining which employee instances in the two databases are the same is a complicated task if they do not share the same key. Entity identification[20] is the process of determining the correspondence between object instances from more than one database. Combining data instances involves entity identification (determining which instances are the same), and resolving attribute value conflicts (two different attribute values for the same attribute). Common methods for determining entity equivalence include:

- **key equivalence** - assumes a common key
- **user specified equivalence** - user performs matching
- **probabilistic key equivalence** - only use portion of the key, may make matching errors

- **probabilistic attribute equivalence** - use all of the common attributes
- **heuristic rules** - knowledge-based approach

Data integration is further complicated because attribute values in different databases may disagree or be range values. For example, in parametric data, the data is spread out in space, and data values vary from one point in space to another (temporal data is a special case of parametric data). Integrating such data may be arbitrarily complex.

Some work on data integration includes Lim's work on entity identification [20], using evidential reasoning to resolve attribute conflicts [21], and combining data in parametric databases [11].

A very good method for data integration appeared in [27]. In this work, the authors attached context information to simple data values. For example, context information on a stock price including currency value and scaling factor. This context information was stored using defined names and values, so that comparisons between data values under different contexts was possible. A set of conversion functions were defined to convert from one context to another. For example, a conversion function to convert a currency between U.S. dollars and Canadian dollars could be defined. Using a form of SQL, called Context-SQL (C-SQL), these conversions could be automatically applied transparenly within a query or as requested by the user within the query. This work is very solid work on data integration and can be incorporated into any algorithm which implements schema integration.

2.3 A Schema Integration Taxonomy

In order to classify schema integration methods, we have developed a schema integration taxonomy. The taxonomy shown in Figure 2 has 3 desirable properties.

The first desirable property of a schema integration method is the types of conflicts resolved.

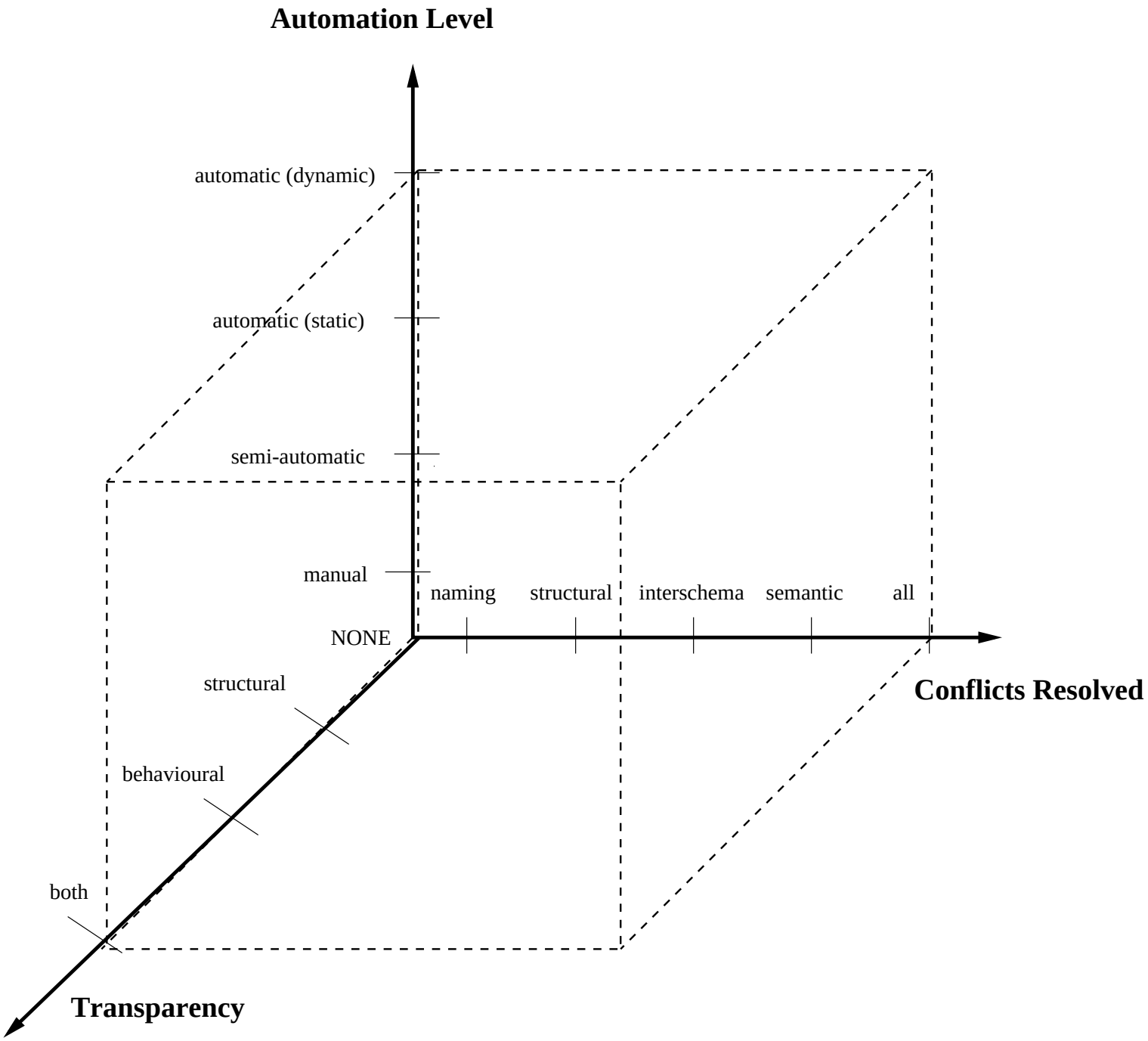

Figure 2: Fundamental Integration Taxonomy

Ideally, a schema integration method should resolve all types of conflicts including naming conflicts, structural conflicts, and semantic conflicts. The power of the integration method is measured by the types of conflicts that it can resolve and is a measure of system applicability. Each method will resolve, or assume away, some or all of these conflicts.

Transparency is a key distinguishing property of schema integration techniques. Transparency is a measure of how the integration is hidden from the user and the application. Systems which do not produce a global view generally have poor transparency. There are two keys forms of transparency: structural transparency and behavioral transparency. A system which provides structural transparency hides the integration, distribution, and organization characteristics of the MDBS. That is, these systems hide the data structure from the user and application who accesses the data at a higher conceptual level. Most centralized databases provide good structural transparency as the user is hidden from most data organization characteristics. In a MDBS, the user must also be hidden from distribution and integration/structural conflicts to provide true structural transparency. Behavioral transparency is a measure of the systems ability to hide database operational characteristics from the user. In a centralized database, the scheduling of transactions is hidden from the user and can be considered a form of behavioral transparency. In a MDBS, behavioral transparency also includes masking distributed updates, handling replication and migration, and updating component database systems using global transactions. Behavioral transparency is much harder to achieve as it relies on a system for executing global transaction updates and propagating these updates to the local databases as needed. Basically, transparency is a measure of the system implementation difficulty.

The automation level measures the power of the integration procedure and its ability to be automated. It is a good measure of system performance. Automation level is also a good distinguishing characteristic because many algorithms that perform schema integration

are not suitable for the computer environment. Heuristic algorithms and algorithms that rely on user/programmer support have poor automation characteristics and suffer from complexity concerns and time-consuming integration procedures. A good integration algorithm will resolve conflicts transparently in an automated manner. Having a designer resolve conflicts is both costly and prone to error.

The three characteristics of resolved conflicts, transparency, and automation level form a fundamental integration methodology taxonomy. However, there are various other distinguishing characteristics among these algorithms. A finer-grained taxonomy involves 8 characteristics divided into 3 taxonomy levels. The 3 taxonomy levels are:

- The integration inputs
- The integration methodology
- The integration product

The first level of taxonomy characterizes methods based on their inputs (see Figure 3). The integration sources axis distinguishes what types of data representations the integration algorithms was designed to handle. Different integration sources including legacy applications, relational databases, object-oriented databases, or a diverse mixture of data sources. Ideally, an algorithm should be able to handle a diverse mixture of data sources, but some algorithms will only target a particular data source and then rely on schema translation or mapping to convert other data sources into a standard form. The two other classification categories are metadata content and metadata structure. Metadata content is the type of metadata stored and used by the algorithm including structural and operational metadata. The metadata structure categorizes how this metadata is stored and represented by the system. These categories differentiate techniques which use metadata from those that do not. Metadata source, referring to how metadata is gathered by the system, could be another possible axis. However, it was not

included because most techniques are currently manual (does not serve as a good classifier), and it is more of a integration methodology category than an integration input category.

The second level of taxonomy focuses on how the integration is achieved (see Figure 4). The integration methodology level classifies techniques based on their implementation characteristics, automation level, and types of conflicts resolved. Both the automation level and resolved conflicts axises are considered "fundamental characteristics" and were explained previously. The integration methodology axis is how the integration is achieved at the implementation level. Integration could be performed using a programming interface, heuristic algorithms, schema transformations, logical rules, or a complex combination of techniques. The implementation technique is important because two differently implemented algorithms may have widely different performance (based on their implementation) even though they demonstrate the same conflict resolution, transparency, and automation level.

Another category considered for this taxonomy was integration procedure which defines how the schemas are combined (eg. one at a time, binary, all at once, etc.) This was not included because it has limited distinguishing properties except for early heuristic algorithms. Most techniques attempt to combine schemas on an as needed basis and not all at once.

The integration methodology category is intimately connected with the other classification categories. For example, if the integration methodology is a programming interface, this naturally implies that the automation level is manual and that the types of conflicts resolved are dependent on the programmer. Also, the system will demonstrate no transparency. However, these same characteristics may be duplicated using a logical rules implementation. The performance of these two systems, although they demonstrate the same features, is connected to their implementation technique. In this case, the programming interface will probably be faster than the logical rules implementation, even though it is harder to implement.

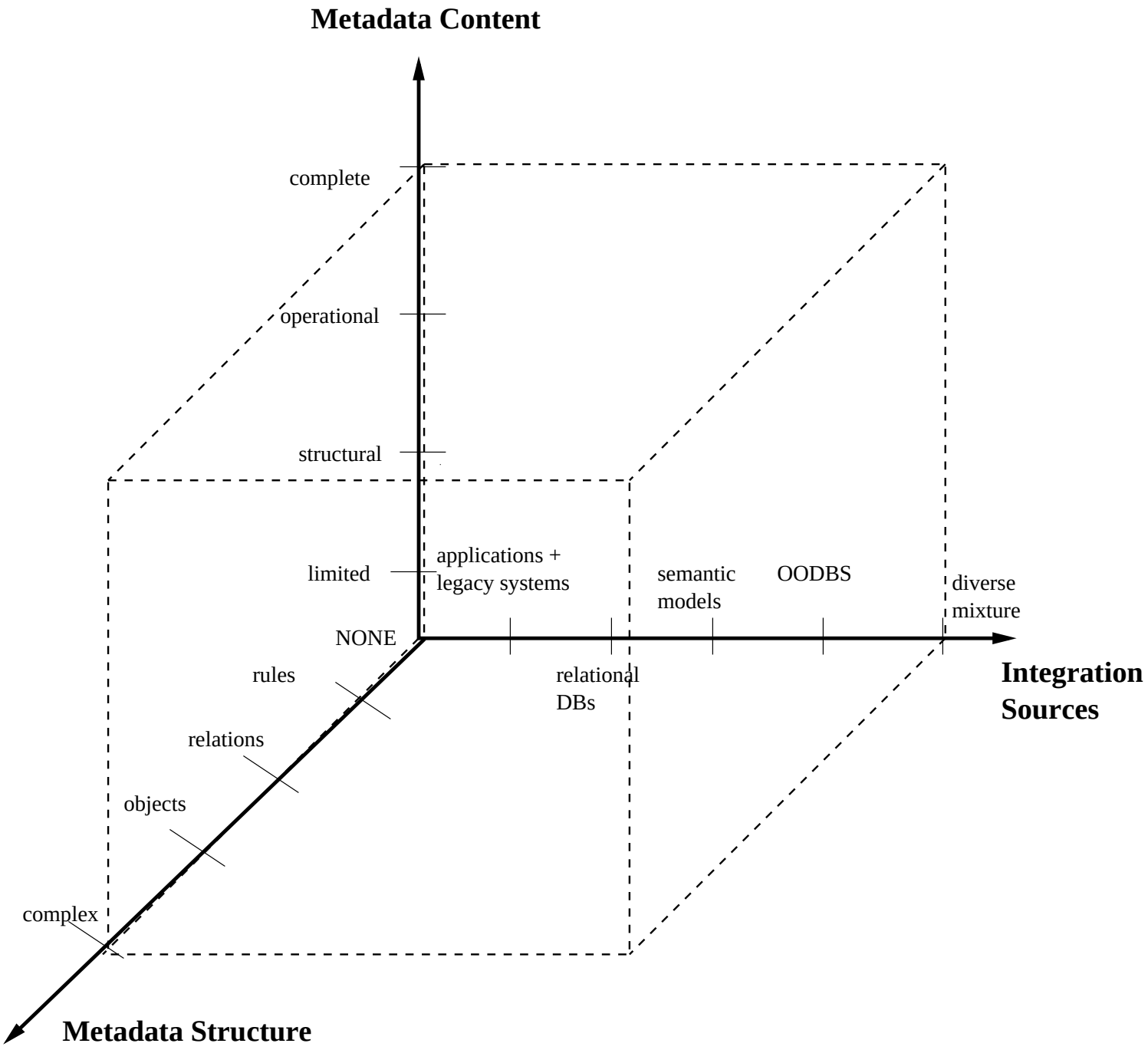

Figure 3: Integration Inputs Taxonomy

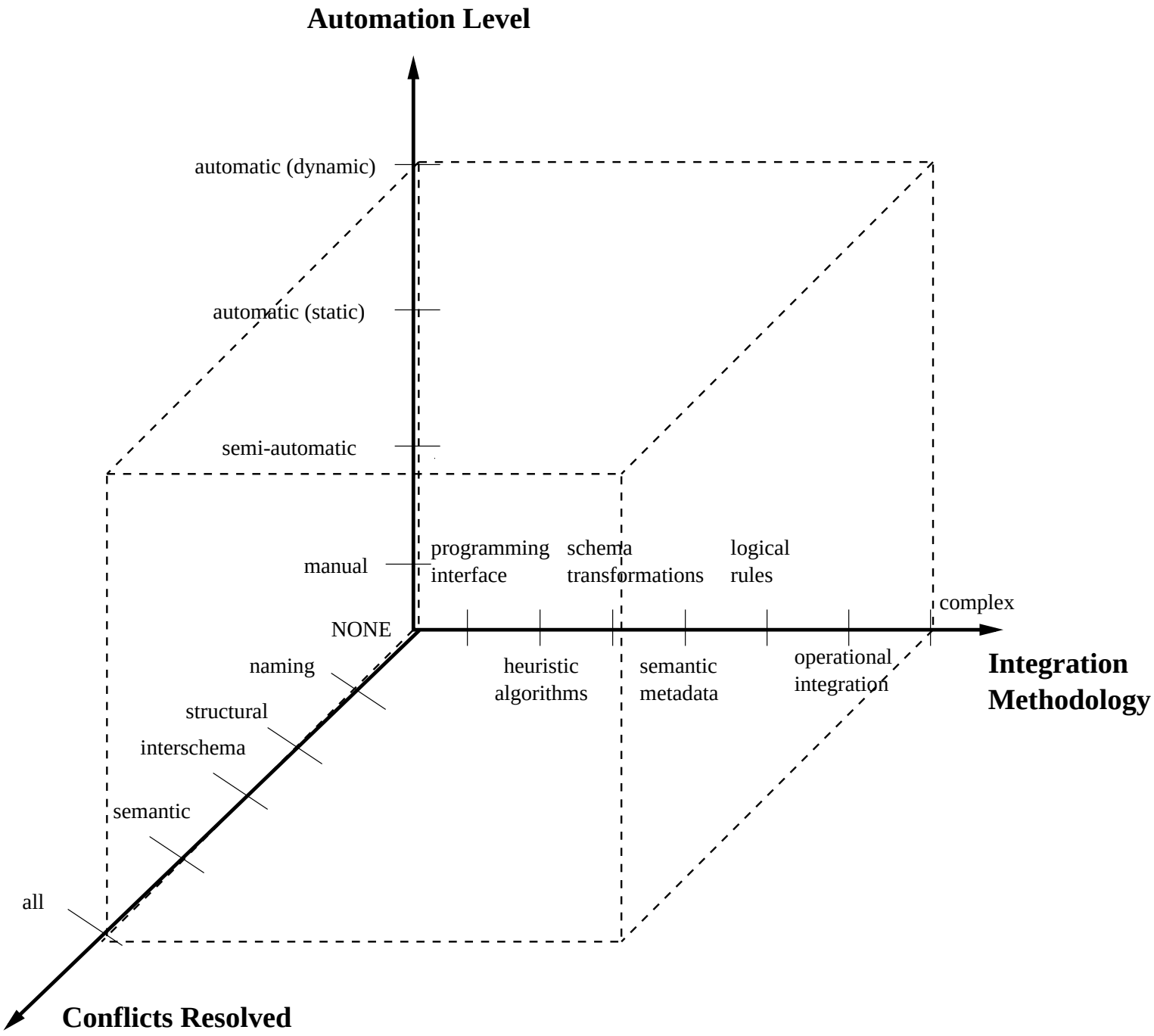

Figure 4: Integration Methodology Taxonomy

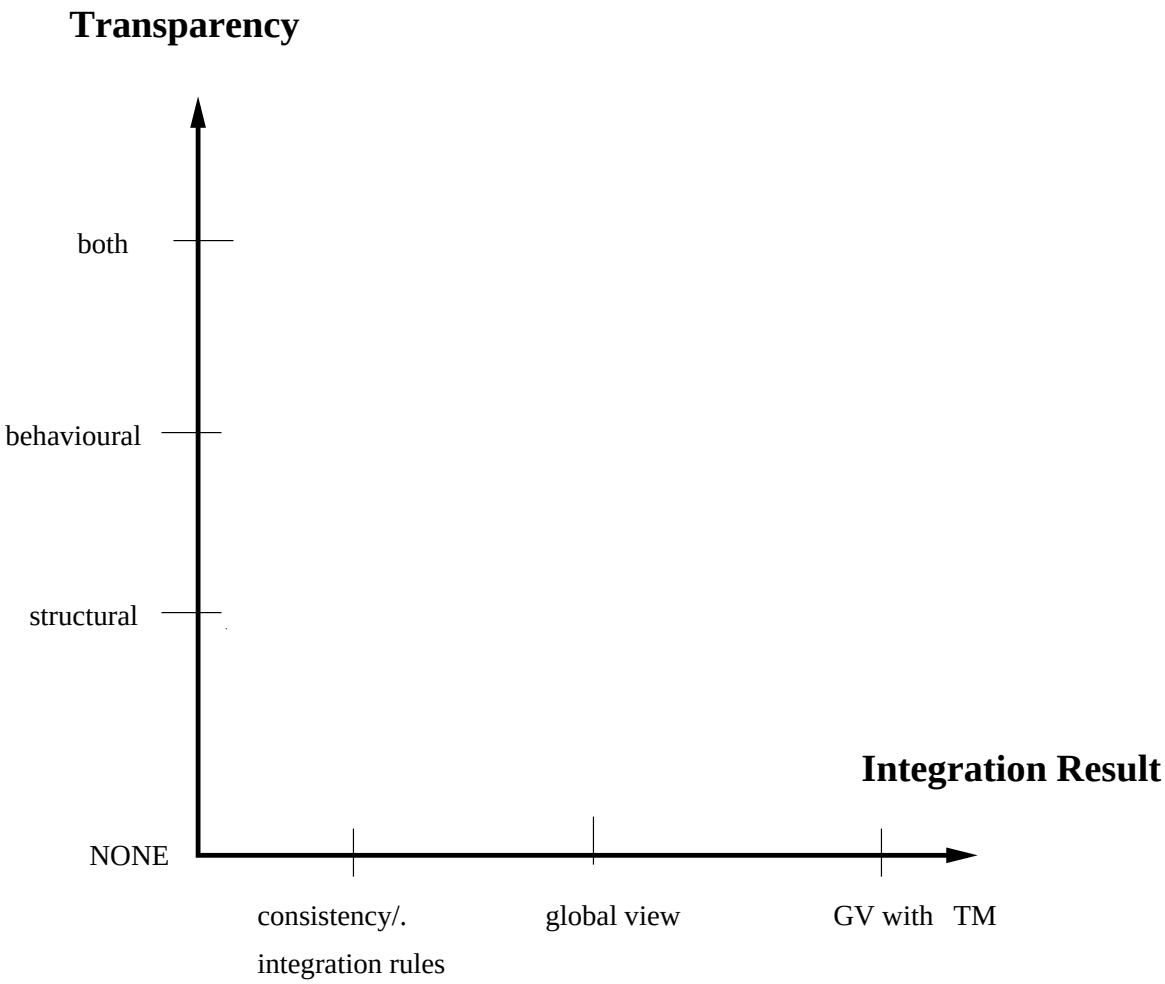

Figure 5: Integration Product Taxonomy

The integration product, see Figure 5, is the third level of taxonomy and classifies the end result after the integration has been performed and its usefulness to the user. Most integration methodologies are designed to provide a consistent, global view, but other methodologies may go further and provide transaction management and operational characteristics and rules. The transparency of the system is a measure of how the integration is hidden from the user and the application.

The integration result axis defines the result of integration. Most systems produce a global view, although other systems also provide a form of transaction management to the integrated

systems. Systems which rely on a programming interface or consistency rules enforce consistency as needed and do not provide a logical, global view of the data. Thus, these systems suffer from poor transparency and often do not even hide the programmer or user from structural conflicts (structural transparency). The transparency axis is a fundamental axis which has been discussed previously.

Consistency and integration rules are considered less desirable than a global view in this taxonomy because they are often context-dependent and integration specific. Moreover, they are often rules relating local database sources to each other, instead of relating local database sources to a global view. Without a mapping to some form of global view, the rules become dependent not only on changes to the local databases, but to other databases referenced by the rules. These rules do not scale well when data sources are added or removed from the system, and do not provide the programmer or application with a complete, coherent view. Rather, the "global view" must be extracted by using the rules to achieve mappings between the diverse data sources.

In summary, the 8 characteristics divided among the 3 taxonomies represent a finer classification of integration protocols. In addition, all integration models should also have the property of high expressiveness to insure that they can capture complex representations of all data models in some form. These characteristics are not all in the "fundamental taxonomy" because many are dependent on each other and are weak classifiers. For example, the integration methodology category is an implementation feature whose classification power is only based on how the results are achieved. Similarly, although the integration sources and integration results categories differentiate algorithms based on their integration product and integration inputs, they have little classification power. Finally, the metadata content and metadata structure categories are not fundamental because it may be possible that an algorithm could resolve conflicts, automatically,

at a high-level of transparency without using metadata (however unlikely). Thus, the use and structure of metadata is not "fundamental" in a perfect integration algorithm and is not needed in a fundamental taxonomy.

The following sections review some of the proposed solutions to the schema integration problem and discuss how they fit in the taxonomy. These solutions will be categorized as follows:

- **Model-based methods and heuristic algorithms** - including relational, functional, semantic, and canonical models
- **Schema re-engineering/mapping** - using transformations to convert schemas
- **Metadata approaches** - adding data about the data and the schema
- **Object-oriented methods** - using object-oriented models as canonical models
- **Application-level integration** - no automatic integration technique, the applications are responsible for handling the integration themselves
- **Specifying interdatabase dependencies** - integrating databases using integration rules which are applied to maintain consistency
- **AI techniques** - using artificial intelligence to compare schemas and knowledge bases/ontologies to store schema information
- **Lexical semantics** - expressing data/schema semantics in free-form language which is automatically compared during integration

2.4 Model-based methods

The earliest and most common methods of schema integration proposed were based on using semantic models to capture database information. Then, the semantic models were manipulated and interpreted, often with extensive user intervention, to perform the integration.

In 1986, a survey of these model-based methods was performed in [1]. These methods attempted to define procedures and heuristic algorithms for schema integration in the contexts

of database view integration (producing a global view in one database) and database integration (producing a global view from distributed databases). Their goal was to "produce an integrated schema starting from several application views that have been produced independently."[1]

Early pioneers in the integration area were quick to enumerate the potential difficulties in their task. They recognized the problems caused by different user perspectives, using different model constructs, and determining equivalent structures representing an identical concept. The method performed by most algorithms of the time relied on performing the following four steps:

- **Pre-integration** - analyzes schemas before integration to determine the integration technique, order of integration, and collect additional information
- **Schema comparison** - compares concepts and looks for conflicts and interschema properties
- **Conforming the schemas** - resolves schema conflicts
- **Merging and restructuring** - merges and restructures the schemas so they conform to certain criteria

Various proposed algorithms followed these basic steps once or multiple times to produce integrated schemas that were judged on the basis of completeness and correctness, minimality, and understandability. However, most algorithms allowed minimal automation and often relied on the user to manually find and resolve schema conflicts. The inputs to the system consisted of the enterprise view, some database assertions (constraints), processing requirements, and mapping rules from component schemas to an integrated schema.

Early integration techniques could generally be classified in one of two schools: relational/functional models or semantic models. In relational models, integrators made the universal relational assumption (every attribute in the database had a unique name) which allowed them to ignore naming conflicts. Several good algorithms could then be developed using the relational model's set properties. Semantic models dealt more with conflicts and did not assume

certain naming characteristics or design perspectives as in the relational models. Thus, the task of integrating using semantic models was more difficult.

The diversity of models overviewed in [1] was extensive, but one important fact summarized their effectiveness: "..among the methodologies surveyed, none provide an analysis or proof of the completeness of the schema transformation operations from the standpoint of being able to resolve any type of conflict that can arise." Basically, early work on schema integration lead to a good definition of the problems and some heuristic algorithms that humans could apply to integrate diverse schemas in most cases.

Semantic models were also surveyed in depth in [14]. The ER-model is considered a semantic model. Another example of a semantic model is a semantic network which represents real-world knowledge in a graph-like structure. Semantic models have rich techniques for representing data, but were tailored for human use in database design and not specifically for automated database integration.

Early in their existence, semantic models were compared to object-oriented models and were deemed better than object-oriented models because they had richer type constructors, and the inheritance of attributes in semantic models was easier than the inheritance of methods in object-oriented models. Semantic models were also considered better than relational models because they achieved better separation between the conceptual and physical views of the data, had decreased semantic overloading of relationship types, and had more convenient abstraction mechanisms. Although semantic data models were good for describing the data, they were insufficient for use during integration because they were often too complex and were not designed for the integration task.

More recent model-based methods generally are based on defining a canonical model for representing the global view. Then, the schemas of the component databases could be mapped

into the canonical model and integrated. A recent survey on the suitability of data models as canonical models was done in [26] and presents a taxonomy of desirable features a canonical model should have. Their findings state that object-oriented models are better canonical models than ER-models (semantic models) because of their expressiveness and ability to capture additional semantic information not already in component schemas. There is an ongoing debate on what models serve as the best canonical models.

Other early work in the area included the work by Hull in [13] in determining the relative information capacity of relational database schemas. An important result from this work is that even if a similar query can be posed on two schema, this does not imply that the schemas have a semantic connection between them. The behavior of objects was studied by Brodie in [5]. In this work, he modeled the behavior of objects using graphical notation. Modeling object and data behavior adds additional complexity to schema integration.

Foundation work in this area inlcudes work on SDM [12] and DAPLEX [30]. DAPLEX was a data definition and query language based on the functional data model. In this model, functions are defined to describe the database. Queries are formulated by combining functions defined on the database to return the desired results. DAPLEX allows very general functions to be defined included aggregation functions, functions representing subtype/supertype relationships, and functions defining derived data and conceptual abstractions. Such mechanisms allow the system to support different user views and manipulate metadata. This generality could allow DAPLEX to be used as a front-end to existing databases and act as a universal query language. The reason why DAPLEX could not achieve such lofty goals resides in the naming and structural conflicts which occur between two databases. Although DAPLEX provides the ability to model a given database using the functional model, combining two different databases expressed in the DAPLEX model is still complex because naming and structural conflicts must be resolved

between the two models. It then becomes necessary to develop specialized functions which perform the mapping in between the two different data sources. Although this mapping is achievable using the DAPLEX language, it is just as complex to construct as customized programming techniques and must be applied between all databases participating in the MDBS.

The Semantic Data Model (SDM) [12] is foundation work on capturing data semantics. In this model, a database is defined as a collection of entities which are organized into classes. A class is a meaningful collection of entities. Database classes are logically related by interclass connections which are similar to relationships in other models. The ability to define complex interclass connections and associate a semantic meaning with their definition is the property that gives SDM its descriptive power. Finally, attributes are defined on both classes and entities.

As mentioned previously, the ability to define complex interclass connections allows SDM more semantic descriptive power. Interclass connections may be defined as subclass connections relating two classes by a subclass/superclass relationship or by grouping connections which relate classes based on grouping conditions which may be based on attribute values or entity presence in other classes. Attributes in different classes can be related by defining attribute interrelationships and can be accessed from different based classes using mappings based on these interrelationships. Thus, a SDM schema consists of base classes of "real-world" entities augmented with derived classes using interclass connections. Attributes from different classes can be accessed using the defined mappings between attributes and interclass relationships. This structure allows SDM to capture more semantic data about the database during its construction.

The major weakness in SDM is the difficulty in reconciling name differences. SDM allows more semantic data on a given data source to be collected which is very useful during the integration. However, the derived classes often have very complex names which are difficult to reconcile with other derived classes. SDM's ability to capture the relationship between classes

explicitly and to capture attribute semantics and mappings between attributes and interclass connections provides a good foundation for an integration model, but lacks a model formulation which is more suitable for automatic integration.

The work on semantic and canonical models has not lead to any practical algorithms for schema integration. Although schema information can be represented in these models, and they can be used to perform the integration manually or using heuristic algorithms, they are insufficient for automating the integration process.

In terms of the taxonomy, heuristic algorithms based on semantic models offer low transparency and low automation although they can resolve most structural conflicts. Behavioral conflicts are typically not considered. These algorithms are applied by designers during integration to construct a global view. Any structural or semantic conflicts that arise must be detected and resolved by the designer. Hence, these algorithms offer low transparency during the design phase as the designer must recognize and resolve all conflicts. However, once the global view has been created, the operational system demonstrates high transparency as queries are mapped through the global view. As the algorithms must be applied by designers, these systems have poor automation characteristics which make them undesirable for large, complex integration tasks. Thus, additional work has been done in the field of model transformations with the hope that the ability to transform between data models will allow more accurate model comparisons and simplified schema integration.

2.5 Schema re-engineering/mapping

A logical extension of the semantic modeling idea is schema re-engineering. In schema re-engineering, schemas to be integrated are mapped into one canonical model or simply mapped into the same model. Then, the schemas are "compared" by performing semantic preserving

transformations on them until the schemas are identical or similar in respect to common concepts. By automating the mapping process and providing a set of suitable transformations, the goal is to compare diverse schemas for similarities and merge them into one schema.

There have been numerous approaches based on schema transformations. One approach [16] constructed a generic mapping tool which was able to map schemas into a meta model. The meta model was a model capable of describing all data models. Then, the schema could be mapped from the meta model to the target model. Their work successfully mapped from a relational schema to an ER-diagram with minimal user input. Work in [24] used a graph structure to represent a database and its constraints to define a set of equivalence preserving transformations. However, the transformations could not be applied automatically.

An in-depth study of schema transformations in an OODBS was presented in [32]. Transformations included renaming, aggregation, and objectify. From the many possible transformations and equivalent schema, the target schema was chosen based on the relatedness of the object methods in the schema. Various measures quantified the object methods correspondence with object attributes and were used to define a heuristic algorithm. The heuristic algorithm was too complex and open-ended to be of use in an automated system.

In total, schema re-engineering and mapping methods yield good results when mapping from one model to another, and at the very least, provide techniques for mapping diverse schemas into the same model. Unfortunately, comparisons on schemas mapped to a canonical model are still not possible because of the many possible resulting schemas after the mapping has been completed. Furthermore, the set of equivalence preserving transformations defined is not sufficient to determine if two schemas in the same model are identical. Without the ability to define equivalence, mapping into the same data model does not solve the integration problem, but rather transforms it into the complicated, and no simpler, problem of determining schema

equivalence.

Schema re-engineering techniques suffer from the same problems as the heuristic, semantic model algorithms on which they are based. Mapping between schema representation models is not a solution to the schema integration problem by itself. The models must map into a schema representation model which allows equivalence comparisons. Thus, these systems resolve no conflicts by themselves, but rather may transform diverse schemas into a single, more usable model. The transformation of schemas is automatic in nature, but no conflicts are resolved and no transparency is provided.

2.6 Metadata approaches

Schemas specified in legacy systems are hard to integrate because there is no metadata describing the data's use and semantics. Metadata approaches attempt to solve this problem by defining models which capture metadata. Then, the user or the system, can use schema metadata to simplify the integration task.

In [31], metadata is a set of rules on attributes. A rule captures the semantics of an attribute both at the schema level and the instance level. Then, an application could use these rules to determine schema equivalence. This approach is good because it captures some data semantics in the form of machine-processable rules. Unfortunately, the rules must be constructed by the system designer, and the actual integration of the rules must be performed at the application-level. That is, the system designer constructs the integration rules, and the applications, using the global system, consult these rules at run-time to provide the necessary schema integration. Thus, applications are not isolated from changes in the semantics or organization of the data, which is the fundamental reason why applications use database systems.

There are several other approaches which capture metadata in some manner. However, they

all suffer from the fundamental problem of failing to use this captured metadata appropriately. There has been no general solution for capturing metadata in a model such that it can be used to automaticly integrate diverse schemas without application or user assistance.

Metadata approaches are diverse in terms of the metadata content and the structure in which it is used. However, most systems typically represent metadata using logical rules which can then be used by the application at run-time to determine data equivalence. These rules can be arbitrarily complex and will often resolve most semantic conflicts. This also provides a limited form of transparency as the application is less responsible for the integration as it can use the previously defined integration rules. In total, these systems tend to resolve most structural and semantic conflicts, with low transparency, and low automation. These systems have low automation characteristics because defining applications which use these integration rules appropriately is difficult and may be subject to changes in the underlying databases or rule definitions. Metadata approaches based on rules are a combination of application-level integration with a little knowledge-base information added to ease the programming task.

2.7 Object-oriented methods

The object-oriented model has grown in popularity because of its simplicity in representing complicated concepts in an organized manner. Using encapsulation, methods, and inheritance, it is possible to define almost any type of data representation. Consequently, object-oriented models have been used not only to model the data, but also to model the data models. That is, object-oriented models have seen increased use as canonical models into which all other data models are transformed into and compared.

The object-oriented model has very high expressiveness and is able to represent all common modeling techniques present in other data models. Hence, it is a natural choice for a canonical

model. The OODBMS transformations previously discussed in [32] focused on comparing schemas in the OO-model. There has also been work on defining a logical semantics for object-oriented databases [23]. It is also possible to automatically generate an OODBS schema from an abstract specification in a high-level language [2].

Although this work is a promising first step toward integration using an object-oriented model, the work is still missing a definition of equivalence. Without a definition of equivalence, it is impossible to integrate two schemas, regardless of the power of the schema transformations. The ability to generate an OODBS schema from a high-level language is promising because it may be possible to define equivalence using the high-level language instead of at the schema level. This may eliminate some of the difficulties in defining schema equivalence.

The use of an object-oriented model as a canonical model for integration is possible. However, the object-oriented model is very general which makes determining equivalence between schemas quite complex. Current techniques based on object-oriented models are in their infancy and tend to rely on heuristic algorithms similar to semantic models. Thus, most conflict resolution is done manually resulting in low transparency and low automation.

2.8 Application-Level Integration

In application-level integration, applications are responsible for performing the integration. This eliminates the need to perform schema integration at the database level and gives the application more freedom when accessing different data resources. However, applications become more responsible for integration conflicts and no longer are hidden from the complexities of data organization and distribution. Although applications may be able to integrate different data sources easier in certain situations, the applications are no longer independent of data organization or format and become arbitrarily complex. Thus, although application-level

integration may have benefits in certain situations, it is not a generally applicable methodology.

A variation of application-level integration is integration at the language level. In these systems, applications are developed using a language which masks some of the complexities in the distributed MDBS environment. Many of these systems are based on a form of SQL.

The first step in constructing language-based integration systems is determining the types of conflicts that can arise. Typically, semantic conflicts are left to the application programmer to determine. Structural conflicts inherent in the data organization however are captured. A good survey of structural conflicts in relational SQL was done by Kim [18]. In this work, structural conflicts are categorized into schema conflicts and data conflicts. Schema conflicts include representational conflicts such as handling when a data item is represented as a table in one schema and an attribute in another. Schema conflicts also include naming conflicts, constraint conflicts, and cardinality conflicts. Data conflicts include missing or wrong data or different representations of data (data format, units, etc.). How these conflicts are resolved by the language is very important in determining its usefulness.

In Bright's survey of MDBS issues [4], he differentiated schema level conflicts from language level conflicts. Schema level conflicts arise from trying to construct a coherent, global view from many data sources. As the number of data sources increases, resolving all semantic conflicts may prove impossible. To combat this scalability issue, he proposed language level integration where a language library helps the user and local DBA perform the integration. Although this may be more scalable, it suffers from poor automation and transparency.

One of the greatest supporters for MDBS integration at the language level has been Litwin. In [19], he proposes a language method for querying and updating in a MDBS. The language is called IDL (Interoperable Database Language), and has interoperability features that subsume those of MSQL. The language allows the user to define higher order queries and views by

allowing database variables to range over metadata in addition to regular data. Metadata includes database names, relational names, and attribute names. This allows the language to construct queries across database systems in addition to regular relational expressions.

Integration is achieved by defining higher order views which can resolve some of the naming and structural conflicts between databases. A high order view is defined using rule-based expressions which can range over data and metadata. MDBS updates can be defined using similar rule-like expressions called update programs. Although defining views over a language provides some integration capabilities, these views have to be constructed manually by the schema administrator and are subject to change. As more databases are added, constructing these views is difficult, and there is little transparency as the schema administrator must be aware of the semantics of relations and attributes at all sites to be integrated. Thus, although the language provides facilities for an administrator to integrate MDBS data sources, the amount of manual work and lack of transparency and automation makes this method no more desirable than the early heuristic algorithms.

2.9 Specifying Interdatabase Dependencies

Enforcing consistency across multiple autonomous databases is very difficult and costly especially when all updates and transactions are atomic. An approach recommended by Sheth [25] involves specifying interdatabase dependencies and enforcing them at certain times. Instead of providing immediate consistency as in typical databases, consistency can be delayed enforced dependent on a certain period of time as required by the application and users. Consistency can also be enforced depending on the data characteristics such as percentage of inconsistent tuples, or functions based on changing data values. Finally, consistency enforcement can be linked to operational features. In this case, consistency may be enforced after an update, before

an access, after a given number of access, or after a certain operation or transaction. Thus, consistency between databases can be enforced as needed which reduces the burden placed on the MDBS.

In this system, interdatabase dependencies are specified using data dependency descriptors (D^3). Where a data dependency descriptor is a five-tuple defined as: $D^3 =< S, U, P, C, A >$

- S - set of source data objects
- U - target data object
- P - Boolean-valued interdatabase dependency predicate defining the relationship between S and U (True if that relationship is satisfied)
- C - Boolean-valued mutual consistency predicate that specifies consistency requirements and defines when P must be satisfied
- A - collection of consistency restoration procedures specifying actions to be taken to restore consistency and to ensure that P is satisfied

Basically, these systems rely on dependency systems at each local site to store and execute procedures implied by these rules. In this manner, each dependency system must be aware of all transactions submitted to its local database because it may have to enforce consistency based on the rules depending on a transaction's actions.

Although the ability to relax consistency constraints has numerous applications in industry, specifying interdatabase dependencies has several problems. First of all, specifying the dependencies is very complex and must be done at the object level. Each dependency between all objects effecting a given object must be specified in rules and update procedures written for them. Thus, mappings will grow as the number of databases grow because mappings are between local databases and not directed through any form of global view. Thus, scalability will become an issue.

In addition, the update procedures are essentially application coded consistency. It has long

been felt that this type of custom programming to resolve consistency problems is undesirable because it is complex, time-consuming, and subject to underlying database changes. Also, it may not be a simple task of intercepting all transactions to a database for the dependency system. Many older legacy applications do not have clear transaction management rules and the dependency system may have restricted access to the internal workings of these systems.

A more fundamental problem is that relaxing consistency introduces a new level of semantic conflicts. With relaxed consistency, an application only is assured of consistency based on the rules. However, application and database users may have come to expect the most current information when using the program. Also, resolving conflicts between consistency requirements between applications using the same data may be complex. One application may expect an object be current for the last 24 hours, while another may expect the object be always current. Resolving these types of semantic timing conflicts is tricky especially when the number of applications sharing and updating the objects grow.

Despite its impressive specification, specifying interdatabase dependencies is essentially a formalized method for coding integration patches between systems. Although relaxed consistency may be beneficial to certain applications, the rest of the system exhibits poor integration properties. According to the taxonomy, the automation level and transparency are low because the programmer or designer is responsible for manually detecting and coding integration rules to resolve conflicts. No global view is produced, and the system does not specify a mechanism for executing global queries and updates. In total, specifying interdatabase dependencies is good for enforcing relaxed consistency but is no better an integration methodology than customized, application-level integration programming.

2.10 AI Techniques

Reasoning with incomplete and inconsistent data has long been a focus of artificial intelligence research. When two schemas with limited data semantics are combined, the problem contains both incomplete and inconsistent data. Thus it is natural that AI techniques have been applied to the integration problem with varying success.

The major reasoning in AI techniques is that the fundamental database model is insufficient when dealing with diverse information sources. In the Pegasus project[17] at HP, databases are combined into "spheres of knowledge". A sphere of knowledge is data, which may be spread across different databases, that is coherent, correct, and integrated. Thus, differences in data representation or data values only occur when accessing different spheres of knowledge.

Another AI approach consists of storing data and knowledge in packets[33]. A global thesaurus maintains a common dictionary of terms and actively works with the user to formulate queries. Each LDBS has its structural and operational semantics captured in an OO domain model and is considered a knowledge source. Users access information in different LDBSs (knowledge sources) by posing a query to the global thesaurus. Query results are posted on a flexible, opportunistic, blackboard architecture from which all knowledge sources access, send and receive query information and results. This blackboard architecture shows promise because it also tackles the operational (transaction management) issue in MDBSs. The only difficulty with the system is the complexity of creating the system and defining how the global thesaurus cooperates with the user to formulate queries.

A very good integration system based on a knowledge base was developed for the Carnot project [8]. In this system, each component system is integrated into a global schema defined using the Cyc knowledge base. This knowledge base (global schema) contains 50,000 items including entities, relationship, and knowledge of data models. Thus, any information to be

integrated into the global schema can either map to an existing entry in the knowledge base or be added to it.

Resource integration is achieved by separate mappings between each information resource and the global schema. Since the system maps to a global schema, these mappings can be done individually and as needed. Each new resource is independently compared and integrated into the global schema. The system also uses schema knowledge (data structure, integrity constraints), resource knowledge (designer comments, support systems), and organization knowledge (corporate rules governing use of resource) during the integration process.

During the mapping, both a syntax and a semantic translation is performed. The syntax translation consists of converting the local language to the global context language (GCL). The global schema contains entries about current data models (relational, object-oriented, etc.) which allow the designer to map the local data model into the global context. Semantics translation occurs between two equivalent expression in GCL (have same meaning) using a set of logical equivalences in GCL called articulation axioms.

An articulation axiom is a statement of equivalence between components of two theories. Basically, an axiom is a logical rule which serves as a mapping between two objects. An object in one view can be converted to an object in the other view (and vice versa) by mapping through the axiom.

The integration procedure in the system proceeds in three phases:

- *schema representation* - represents the local schema in global context language (Cyc format)
- *concept matching* - map Cyc concepts to concepts in global schema
- *axiom construction* - an axiom is constructed for each match found

This integration method has several desirable characteristics including:

- Individual mappings - information sources can be included in the global schema one at a time, independent of other sources.
- Global view constructed - the system maps to a global view consistent across all sites which provides good query transparency when accessing sites through the global view.
- Handles schema conflicts - using mappings to GCL, representational (structural) conflicts are resolved by using a common integration language.

In terms of the taxonomy, the Carnot project has many desirable features. The transparency of the system is high as it only requires the designer to map an information source to the global view once. After that all query translation and mapping is handled through the global schema and GCL. This results in a high automation level as resources can be integrated independently, quickly, and are relatively free from changes in other resources. Also, most types of conflicts can be resolved. Structural conflicts are resolved by mapping data model concepts into GCL concepts. Behavioral conflicts are not studied, although a global transaction model was proposed [34].

It would seem that the Carnot project represents a near perfect solution to the integration problem. The only problem is that knowledge base systems, especially one this large, require significant computational resources. For information resources with limited processing capabilities, this may be a problem. Also, it is unclear how all semantic conflicts are resolved. Mapping a local data model into GCL can be used to address structural conflicts, but the authors made no statement on how conflicts within the GCL are handled. For example, one designer could integrate a concept into the global schema using an entity while another could use a relationship. In this case, the concepts are identical, but the integration problem is now at the global level in GCL. Depending on the frequency of these types of conflicts, this may be a serious problem.

A knowledge base is also used in [10] to capture metadata and database semantics. The

knowledge base is organized as a semantic network, and new terms are integrated by determining and merging into the knowledge base the best sub-network that spans the concepts common to the knowledge base and the newly added schema.

An interesting contribution of the work is that the knowledge base is organized into 4 laerys: concept layer, view layer, metadata layer, and the database layer. The authors correcly realize that integrating databases occurs at higher levels (conceptual,view) than metadata and structural organizations. They then organize the knowledge base to capture and link information at all layers, which simplifies the integration task.

Like other knowledge base approaches, the fundamental problem is the imprecision in the knowledge base. Although common concepts are often well integrated by utilizing spanning subnetworks, this is not always guaranteed. Also, the resulting knowledge base does not provide a "unified" global view sufficient for SQL-type querying. Rather, it only supports imprceise queries or can be used as a tool for integrators to construct views over the data.

2.11 Lexical Semantics

Lexical semantics [9] is the study of the meanings of words. Lexical semantics is a very complicated area because much of human speech is ambiguous, open to interpretation, and dependent on context. Nevertheless, lexical semantics may have a role to play in schema integration. The metadata could be described in free-form language. Using these free-form language descriptions and a suitable lexical analysis tool, it may be possible to automate schema integration.

It is our opinion that free-form lexical analysis is too complicated and inmature field to be used in schema integration. This opinion is strengthened by the fact that no integration algorithms based on general lexical semantic techniques have been proposed. It is difficult

enough to parse free-form language let alone determine its semantics and ambiguity. In the future, lexical semantics may ease the integration task, but further study is needed.

However, the use of lexical semantics plays a prominent role in the Summary Schemas Model (SMM) proposed by Bright *et al.* in [3]. The Summary Schemas Model is a combination of adding metadata and intelligent querying based on the semantic meanings of words. This system is not a general lexical semantics parser which takes a free-form description of a query and executes it. Rather, using user supplied words and a global dictionary, the system uses semantic metadata to translate the query into a suitable query for the underlying systems.

The SSM provides automated support for identification of semantically similar entities and processes imprecise user queries. It constructs an abstract view of the data using a hierarchy of summary schemas. Summary schemas at higher levels of the hierarhcy are more abstract, while those at the leaf nodes represent more specific concepts. The target environment for the system is a MDBS language system which provides the integration. Thus, the SSM is more of a user interface method for dealing with diverse data sources than a method for integrating them. The actual integration is still performed at the MDBS language level. However, the SSM could be used on top of another integration algorithm as its support for imprecise queries is easily extendible.

The heart of the SSM is linguistic knowledge stored in an online taxonomy. This taxonomy is a combination of a dictionary and a thesaurus. The authors used an existing taxonomy, the 1965 version of Roget's Thesaurus. The taxonomy contains an entry for each disambiguated defintion of each form from a general lexicon of the English language. Each entry also has a precise definition and semantic links to related terms. Links include synonymy links (similar terms) and hpernymy/hyponymy links (related hierarchially).

Using this taxonomy, a semantic distance between words can be constructed. The seman-

tic distance allows the system to translate user-specified words (imprecise terms) into actual database terms using the hierarchy of summary schemas. A leaf node in the hierarchy contains all the terms defined by a DBA for a local database. Local summary schemas are then combined and abstracted into higher-level summary schemas using hierarchial relationship between words. In total, this system allows the user to specify a query using their own key words, and the system translates the query into the best fit semantically from the information provided in summary schemas by every database in the MDBS.

It is important to note that no schema integration is actually taking place. The summary schemas (even higher level ones) do not represent integrated schemas but rather an overview of the data in the underlying databases summarized into English-language categories and words. The user interface for the query processor can then translate user queries using the summary schemas from English words provided by the user to database terms provided by the DBAs in the summary schemas. However, the use of a global taxonomy for words and user queries based on words and their meanings may have an important role in schema integration.

Related to lexical semantics are integration techniques which construct semantic dictionaries or concept hierarchies to capture knowledge in databases. Similar to knowledge bases and ontologies, semantic dictionaries and concept hierarchies allow database integrators to capture system knowledge and metadata in a more manageable form.

Castano [7] defined concept hierarchies from a set of conceptual (ER) schemas from Italian government databases. Concepts at the top of the hierarchy are more general than lower level concepts. Concepts are connected in the hierarchy by instance-of-links (if an entity is an instance of a given concept), kind-of links (links concepts based on superconcept/subconcept relationship), and part-of links (links concepts related by aggregation relationships).

The concept hierarchies are defined using either an integration based approach, where the

integrator maps data sources to a pre-existing concept hierarchy, or using a discovery-based approach, where concept hierarchies can be incrementally defined one schema at a time by integrating with known concepts.

The use of a concept hierarchy to define similar concepts is a useful idea. By also defining formuals to determine semantic distance between concepts in the hierarchy, it is possible to estimate semantic equivalence across systems. The authors also presented a mechanism for querying the MDBS based on the concept hierarchy and concept properties. Semantic distance calcuations were used to eveolve and create the hierarchy by adding new concepts or grouping concepts appropriately as they are integrated. The technique provides a good way of dynamically integrating data sources using ER descriptions.

The major problem with this approach is that it does not produce a concept hierarcy which can easily be queried. The authors propose querying the MDBS for entities which are "similar" by vritue of having related structural and behavioral properties. Although this may be sufficient in some cases, the majority of industrial applications require more precise querying using a variant of SQL.

Another problem with this approach is that constructing the concept hierarchy in a discovery-based approach requires human intervention and decision making. The hierarchy may have been more useful if the structure and the behaviour of the concepts were removed, and it solely focused on identifying similar concepts. In total, using a concept hierarchy to integrate databases is ean extremely useful insight, but the hierarchy constructed did not allow precises queries essential to most environments.

2.12 Industrial-Level Integration Techniques

Despite the considerable research effort on schema and data integration, current techniques do not apply well to industrial problems. Many of the approaches are not scalable or require so much human intervention that it is easier to perform the integration by hand.

In industrial settings, a more ad hoc approach is often taken. In [28], a system of describing data dependencies is used to insure consistency among databases in a multidatabase system. This approach is based on specifying interdatabase dependencies as discussed previously. It is used in industry because custom coding is simple, even though it is time-consuming and capital intensive. Also, it represents a solution to the transaction management problem as consistency enforcement replaces many global transactions.

Another industrially applicable project is the Interbase system[6]. The Interbase system is a framework for integrating diverse data sources based on a distributed flexible transaction manager which guarantees global consistency. The system codes interfaces to individual data sources and provides a high-level programming language and graphical interface for access to the data sources. The global transaction manager is simple because it guarantees resource ordering, and most of the integration is performed by hand, nevertheless the Interbase system represents a workable architecture for industrial applications. There are numerous other similar projects, but all rely mostly on human coding and manual resolution of integration conflicts.

3 The RIM Model

Previous work on the integration problem has led to several algorithms with satisfactory performance. Except for the Carnot project and related knowledge base/ontology/concept hierarchy techniques, most of these algorithms have poor transparency and automation, and often rely

on the designer to detect and resolve most conflicts. For my thesis work, I will define the Relational Integration Model or RIM. RIM is designed to integrate simple relational schemas with a high degree of transparency and automation while relying on limited support from the designer.

RIM is very similar to the Carnot project is several respects. The most important of which is the use of a global dictionary for defining the global view. Relational schemas are integrated into the global view one at a time using semantic metadata collected on the relational schema automatically and enhanced by the designer. Using these RIM specifications, the system automatically maps the relational schema into the global view and uses the global dictionary to resolve naming conflicts.

Although the Carnot project was able to integrate more than relational schemas, it did not adequately address how all types of conflicts were handled. In RIM, all structural and semantic conflicts are handled by gathering sufficient metadata in the RIM specification (and using the global dictionary) to automatically import relational schemas into the global view.

The Relational Integration Model has several desirable properties including:

- Individual mappings - information sources can be included in the global schema one at a time, independent of other sources
- Global view constructed - the system maps to a global view consistent across all sites which provides good query transparency when accessing sites through the global view
- Handles schema conflicts - using mappings to the RIM specification, representational (structural) conflicts are resolved by using a common integration language

The following sections outline RIM and its characteristics.

3.1 A General Integration Algorithm

This section overviews a general integration algorithm which is built on a global dictionary to resolve naming conflicts. Although this algorithm does not exactly describe RIM, it overviews the concepts and issues involved in its construction.

A general integration algorithm based on a global dictionary has the following basic steps:

1. Determine integration scope
2. Enumerate global concepts with appropriate names
3. Transform application data into integration language
4. Combine application specifications into a global view
5. Refine integrated schema

The first step common to any integration methodology is to determine the scope of the integration. This process includes defining what data sources will be integrated, and what data contained in these sources will be used and/or summarized in the global schema. Also, this process should define a high-level mechanism on how data conflicts are resolved. Basically, this step produces a list of global concepts and entities which will be represented and stored in the global view, and a set of abstract mechanisms for handling conflicts at the data level. This is the organizational view of the data.

Enumerating global concepts is the second step. In this step, global concepts and entities defined for the global schema in the preliminary step are given meaningful names which are to be used throughout the integration process. Unique names for the global view are required for all the entities, attributes, and possibly relationships to be represented. These unique names are then given to the system integrators to be used when representing application data in the integration language. By defining the global names before integration, the metadata gathered from the applications should be mostly free from naming conflicts, as the system integrators

use the global names when capturing the metadata.

The first two steps are mostly performed at a high level of abstraction. These steps would be performed by management and information technology personnel working together to decide what data will be stored in the global view. Once the data in the global view is agreed upon, the lower level job of mapping application data to the global view must be performed.

Given the set of global concepts and entities with their standardized names, application integrators can work independently on mapping application data into the integration language. Although this process may be performed semi-automatically by extracting schema information using schema re-engineering techniques, much of the additional metadata required for the integration language will have to be inserted manually. At the minimum, the integrator may have to convert names used at the local level into the defined global names. When this representation procedure is complete, there is a mapping to and from the integration language for each application.

For some applications, automatically extracting information is not possible. Such applications generally have a poor mechanism for schema definition or may not have a schema. For database systems, schema information can be used to fill in fields in the integration language including local names and sizes, relationships, entities, and some foreign key information. This information can be readily extracted using schema re-engineering techniques discussed previously.

Most of the metadata on the schema will have to be inserted by the designer. However, the metadata only need to concern this local application. The only tie an integrator of a given application has with the rest of the integrators is the standardized global names which should be used in the mapping. Data which must be manually extracted includes operational behavior, data semantics, and relationship semantics. Although this is a complicated task, it is easier to

capture semantics when examining one application at a time then when examining all at once.

When the data for each application has been mapped into the integration language, automatic integration of data using schema equivalence rules is possible. Many of the conflicts which previously had to be resolved manually can now be handled automatically because they have been reduced by using global, standardized names for concepts. Also, the integration language is structured in such a way that all equivalent semantic representations can be detected automatically. After automatic integration of application schemas is completed, a uniform, global view representing the concepts outlined in the preliminary phases is constructed. A data warehouse can then use this global view to generate relations to store summary data.

During integration, mappings are generated from the global view to the individual, local application views. Remember that the application views are stored in the integration language which itself is a mapping of the integration data. Thus, two levels of mapping are needed to convert from the original application view to the global view.

Operational semantics are much harder to integrate than structural and data semantics. Depending on the ultimate use for the global view, resolving conflicting operational semantics may be quite difficult. Thus, this procedure will most likely require user intervention and tuning.

The final phase of integration is refining the integration. In this step, designers can use the application specifications in the integration language to discover new data which could be represented in the global view, add or change concepts in the global view, or tune the system for performance.

3.2 The RIM Architecture

The RIM architecture (see Figure 6) consists of N relational database systems with N RIM specifications combined into M separate global views for M different clients. That is, each relational database system in the MDBS provides a RIM specifications containing semantic information and schema information on the data it provides to the MDBS. Thus, a RIM specification is similar to an export schema in federated systems. Each client participating in the MDBS takes the RIM specification from the databases it wishes to access and combines them to form an integrated global view. Note that the integrated global view of each client may be different and is often only a subset of the overall MDBS global view.

The global view is constructed at each client by combining RIM specifications and resolving conflicts. Conflict resolution is performed automatically using schema transformations and name comparisons using the global dictionary. The integrated view at each client changes as RIM specifications are added or removed from the view.

A global view consists of a dictionary of global names that are present in the global view. Each global name will be in one of 3 states in the RIM model: an entity, a relationship, or an attribute. RIM specificaitons with new global names/concepts have these concepts added as they exist in the RIM specification. For example, if a "person" entity exists in a RIM specification to be integrated, and it does not exist in the global view, then the global concept "person" is added to the global view as an entity. If the global concept exists in a different form in the global view than in the RIM specification, the concept in the RIM specification is mapped into the appropriate form as the global view.

Basically, the global view consists of a set of structured concepts which will be queried by global semantic name using a GUI. Each entity in the global view (and its conceptual name) is displayed to the user. Every attribute and relationship that the entity has in the global view is

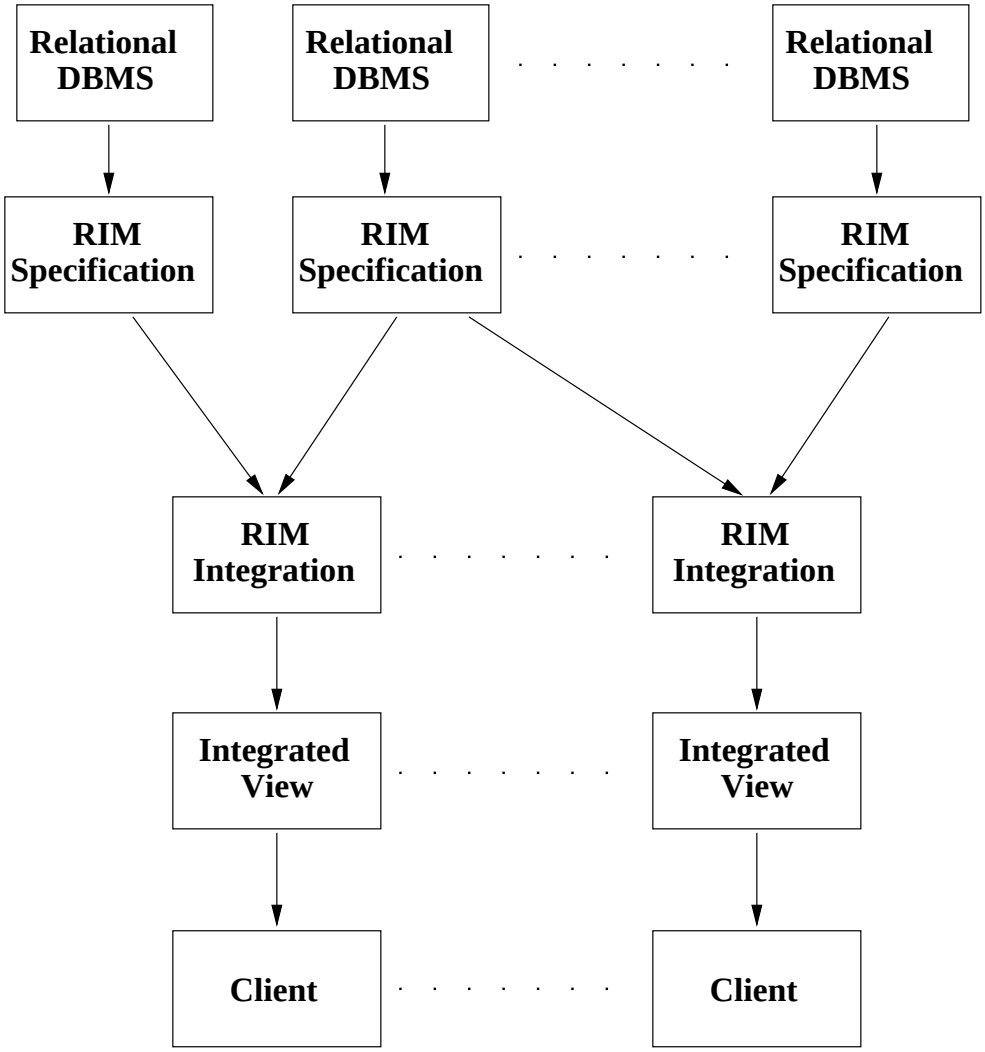

Figure 6: The RIM Architecture

shown under the entity with its semantic name. Also, for each attribute, entity, or relationship, a list of RIM specifications (databases) which contain this concept is shown. The system also stores the form of each global concept in each RIM specification that it exists. For example, in one database, an address may exists as an attribute. If this database is the first database "integrated" into the global view, then the global concept of address is added to the global view as an attribute (presumably of some entity also in the global view). If another database also contains addresses, but these addresses are represented as entities, then the integration process will convert the entity into an attribute, then integrate into the global view. In total, the address concept will be an attribute in the global view, and will be present in DB1 as an attribute and DB2 as an entity.

Queries on the address attribute will be handled by searching for the semantic name address in the specifications for DB1 and DB2. SQL statements querying each database can easily be constructed by using the appropriate field and table names present in each specification. Typically, a user could not query just on addresses because in the global view an address is an attribute. However, since one database (DB2) stores an address as an entity, it would be possible for the user to pose a query just on addresses by only using DB2. An intelligent query processor could then determine the structure of the queries and which database to use, based on the structural form of each global concept in each database.

3.3 The Relational Integration Model (RIM)

The RIM is designed to integrate diverse relational schemas as would be present in conventional relational database systems. There are three basics constructs in the RIM, which is based on the ER-model, but includes additional semantic information to aid integration. The three constructs are:

- **Entity** - is a concept whose existence does not depend on any other entities in the universe of discourse
- **Relationship** - is a combination of two or more entities which does not exist without the definitions of its component entities
- **Attribute** - is a characteristic of an entity or a relation

The RIM format is based on specifying relations in a relational database. RIM assumes all data is stored in two-dimensional tables similar to relations. Each row in a table may represent an entity instance or a relationship instance, and the meaning of a row is explicitly stated in the table header. Each column of a table represents an "attribute" of an entity or relationship instance in some manner. However, a column may not contain an attribute that provides information on the instance, as it may be used as a foreign key for example.

The major assumption in the RIM is that all entities and attributes that are to be in the global view have agreed upon names before integration begins. Then, as system designers integrate relational database systems separately, their final RIM specification will use the globally standardized names.

The RIM specification consists of two components: the table header and the table schema. The table header contains information about a table at the table level including the table name, sharing and access information, table update mechanisms, and information on relationships with other tables. The table schema is very similar to a conventional relational schema. The name, size, and type of each column field is required, but other semantic information including how a field is used is also stored.

3.3.1 The Table Header

The table header provides table-level information about a given table and has fields such as:

- **Name** - unique table name

- **Scope** - where this table is used: system, organization, national, international level
- **Record size** - the size of one record of the table
- **Record count** - the number of records in the table
- **Access mechanism** - stores if the table is read-only, write-only, or read-write
- **Sharing mechanism** - stores what users or systems have access to the table
- **Index list** - a list of the columns (attributes) used as keys for indices on the table
- **Foreign key list** - a list of foreign keys that this table can use to access other tables. An entry in the list contains the key construct (which is a function of table attributes), the table for which the foreign key applies, a user comment about the foreign key, and cardinality constraints associated with the foreign key.
- **Foreign key access list** - a list of tables that have a foreign key to this table. (Entries are similar to the foreign key list.)
- **Record insert mechanism(s)** - methods and frequency of record inserts
- **Record delete mechanism(s)** - methods and frequency of record deletes
- **Record update mechanism(s)** - methods and frequency of record updates
- **Record type** - stores if a table record represents an entity, a relationship instance, or other.
- **Record name** - semantic name for a record in this table. A semantic name is required for a table storing entities, but is not required for a table storing relationship instances, as they may have no meaningful name.
- **Record grouping** - stores the reason why records are grouped together in this table which may include being instances of the same entity/relationship (most common), temporarily related, or no relation.
- **Record distinction** - stores how records are distinguished. This field would store the table key if applicable.
- **Record duplicates** - stores how record duplicates are handled if they are allowed.
- **Table comment** - free-form text comment about the table.

3.3.2 The Table Schema

The table schema in a RIM specification contains information at the column or attribute level of the table. For each column (attribute) in a table the following information is stored:

- **Field name** - the name used by the database system to reference this field
- **Semantic name** - the semantic name of this field whose meaning captures the information it stores. (This field will be used to store the globally standardized names.)
- **Field use** - how the field is used and its size and format. There are several possible choices which have their own individual fields:
  - **attribute field** - describes the record instance more closely or adds additional information to the record instance. This is a very general field and often only applies to character fields like address or phone number. Often, more specific field types should be used to describe the field.
  - **key field** - are fields who have the additional purpose of uniquely identifying the record instance either alone or in some functional combination with other table fields. If a column is a key field, the key's scope and format must be specified.
  - **categorization field** - value categories the entity and changes its characteristics (or how it is handled by the system). For example, a 1-character field may contain character codes which are specially interpreted by applications using the table. Both the codes and their meanings should be specified if possible.
  - **summation field** - stores a functional value computed using attributes in this or other tables. The function must be exactly specified.
  - **date/time field** - stores time information with a given format and size. Unusual formats must have a mechanism for converting into standardized date/time formats.
  - **foreign key field** - stores a key into another table or is used in a functional combination with other table fields as a foreign key. Foreign keys must have a specific format, and state the table(s) for which the foreign key applies.
  - **logical field** - stores boolean valued data. A logical field is a special case of a categorization field, and the user must specify the meaning of true and false.
  - **numeric field** - stores numeric data in a given format and size. The number of decimals must be given, and optionally, a range of valid values.

- **reference field** - references other systems data not using a foreign key mechanism. It is important to recognize such data which provide some integration information already. For example, a log field in an order table may be used to store the key of the order table in another system. This field can then be used to simplify the integration of the two systems.

3.4 Possible Semantic Conflicts in RIM

There are six basic types of semantic conflicts possible in RIM:

- **Attribute-entity conflict** - the same concept is represented as an entity in one schema and an attribute in another
- **Attribute-relationship conflict** - the same concept is represented as an attribute in one schema and a relationship in another
- **Entity-relationship conflict** - the same concept is represented as an entity in one schema and a relationship in another
- **Entity-entity conflict** - two different entities are used in different schema with different degrees of specialization (ie. employee vs. engineer) This conflict is not studied.
- **Attribute-attribute conflict** - two different attribute formats (data formats) are stored to represent the same concept. (Data value conflicts.) This conflict is not studied.
- **Relationship-relationship conflict** - different participation constraints for two relationships representing the same concept and often arises due to different degrees of specialization in the entities involved in the relationship. This conflict is not studied.

3.4.1 The Attribute-Entity Conflict

The attribute-entity conflict occurs when one schema represents a concept as an attribute and another schema represents a concept as an entity. For this discussion, we will only consider single valued attributes.

Two example schemas are shown in Figures 7 and 8. In this example, the two concepts being modeled are people which have addresses. The global standardized names defined are:

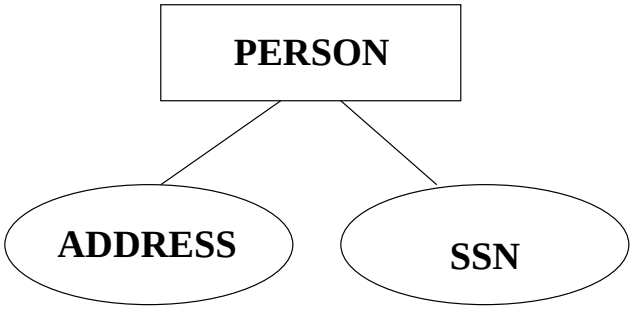

Figure 7: Person-Address Schema 1

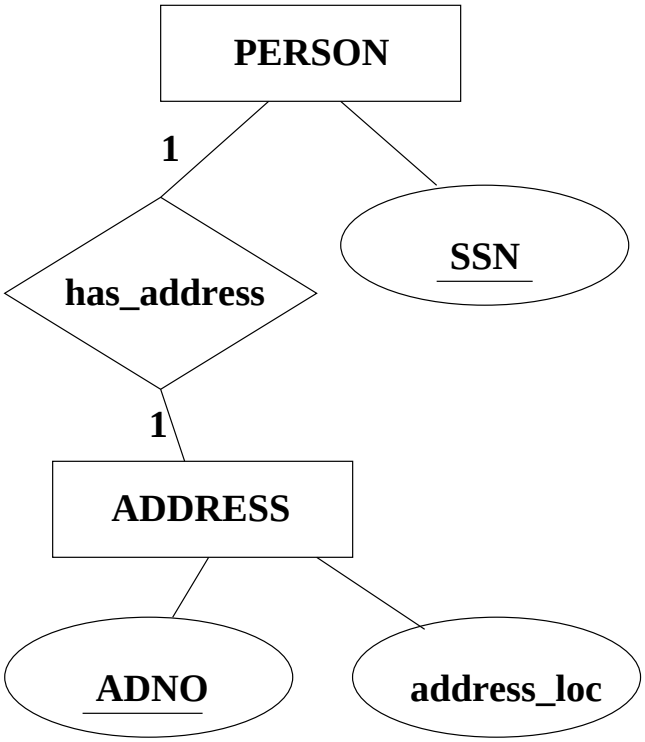

Figure 8: Person-Address Schema 2

person, SSN, and address, and address will be represented as an attribute in the global view. In Figure 7, the address of a person is represented as an attribute of the entity person. If Figure 8, the address of a person is a separate entity which is related to a person by a 1-to-1 relation, as we are only considering single-value attributes.

The relational schema for Figure 7 is: person(SSN,address). The relational schema for Figure 8 is: person(SSN,adno), and address(adno, address_loc). Given these two relational schemas, they must be converted into individual RIM specifications.

For the RIM specifications, the important fields are the record name in the table header, and the semantic names of the attributes and the field usage types of each column of the table. This information must normally be added by the system designer.

The RIM specification for Figure 7 identifies one table which has records of the entity person. That is, the table person stores instances of the concept person which is an entity. The table has fields: SSN (key attribute) and address (general attribute) identified in its table schema.

The RIM specification for Figure 8 identifies two tables. The first table stores records with the semantic name person (entity). This table has fields: SSN(key attribute), and adno(foreign key attribute). In the table header, the foreign key relationship from the person table to the address table is identified. The second table stores records with the semantic name address (entity). This table has fields: adno (key attribute), and address_loc (general attribute).

When integrating these two schemas, the global entity person is identified in both schemas and has the same SSN as the key. However, the global attribute address is not immediately found in schema 2 like it is in schema 1. In schema 2, there is the global attribute name 'address', but it is an entity. Noting that there is a foreign key relationship from the person table to the address table using the adno field, any fields in the address table can be effectively treated as attributes

of the person table. Thus, a person has attributes SSN, adno, and address_loc. Integrating at the data-item level then removes conflicts between the representation of address_loc and address in the two schemas.

It is important to note that schema 2 may store more than person addresses. For example, it may also store business addresses. However, both schemas are semantically equivalent in terms of people and their addresses. Given that we know that we want to map to a global view with a person entity and an address attribute, it was relatively simple to map schema 2 into the appropriate form given the additional information provided in the RIM specification. The only potential difficultly is naming conflicts. For example, the address relation may have be called add_rel in the relational DBMS. These conflicts will be detected by the system designer before the integration process begins. The add_rel relation would be assigned a semantic name of address (corresponding to the global name address) so that naming concerns are not present at the time of integration.

3.4.2 The Attribute-Relationship Conflict

The attribute-relationship conflict occurs when one schema represents a concept as an attribute and another schema represents a concept as a relationship. This conflict occurs when a relation is used to store the values of a multivalued attribute in a relational database. In the following example, the global concepts of shipto (entity) and special instruction (entity) are defined. The shipto entity has the key attribute id and multivalued attributes for special instruction codes and descriptions. The special instruction entity has the key attribute code, and the general attribute description. Figures 9 and 10 show the schemas to be integrated in ER-form.

Neither two schemas are in the identical form as desired in the global view. The problem is that the multivalued attributes desired in the global view cannot be simply represented using

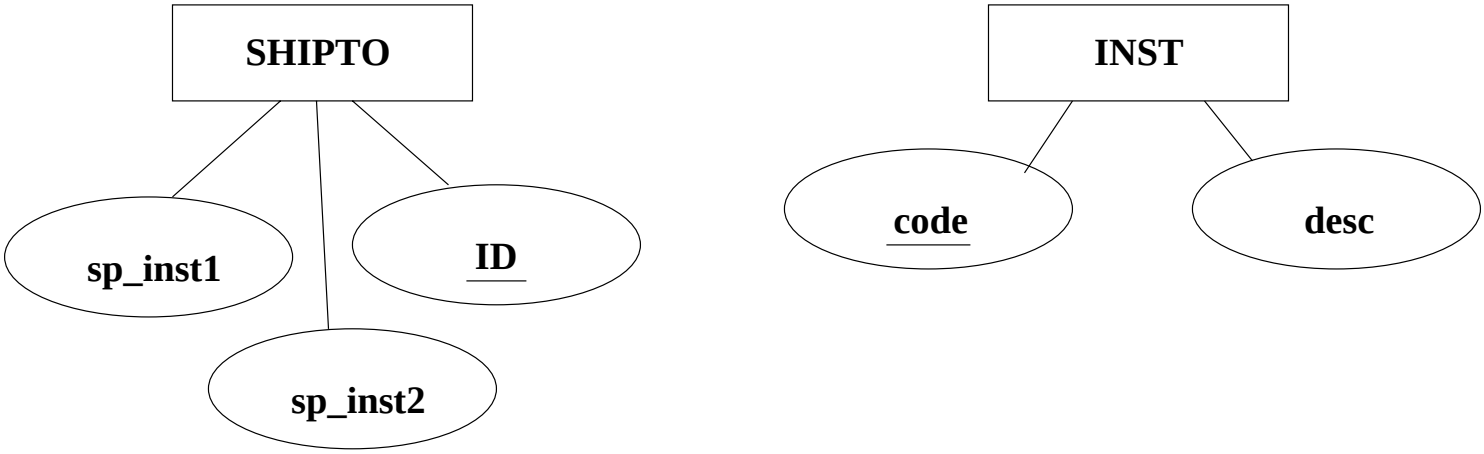

Figure 9: Shipto instructions schema 1

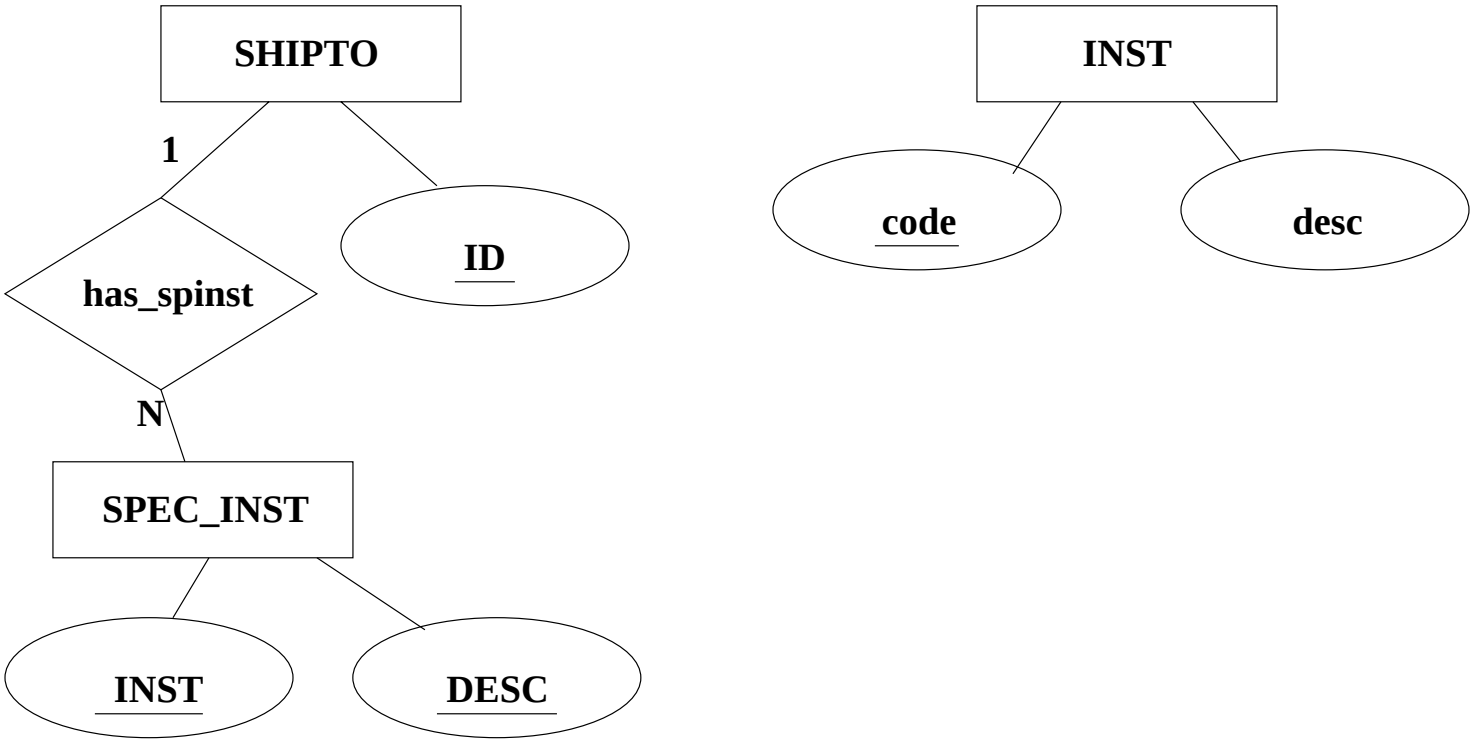

Figure 10: Shipto instructions schema 2

relations. The two representation choices are: using different foreign key attributes of the same entity (schema 1), or a 1-to-N relationship between the two entities (schema 2).

In the RIM specification for schema 1, the shipto entity has key attribute id, and two special instruction foreign keys (sinst1, sinst2) for the special instruction table. Each of these foreign key relationships are 1-to-1, so for each of these foreign keys the attributes of the special instruction table can be considered as attributes of the shipto table. Thus, after this initial mapping, a shipto entity has the attributes id (key), special instruction code and description (1), and special instruction code and description (2). As each of the last two attributes have identical names, the system recognizes them as instances of a multivalued attribute, and redefines the shipto attributes as id (key), special instruction code (multi-valued), and special instruction description (multi-valued) which exactly conforms to the global view schema.

Schema 2 is semantically equivalent to schema 1 even though the multivalued attribute is represented using a relationship. In this case, there is a third table called Spec_Inst which is used to store the instances of the multivalued special instruction field. The procedure for converting this schema into the global schema is as follows. First, the RIM specification contains two entities (shipto and inst (renamed as special instruction using global names)), and a relationship spec_inst. The system is looking for special instruction codes and descriptions as multivalued attributes of the entity shipto. The entity shipto currently has no attributes. However, the special instruction entity does exist (as recognized by its global name in the RIM specification), so there may be a relationship which links the two entities. In this case, the relationship is spec_inst which has a N-to-1 relationship with shipto using id as a foreign key, and has a 1-to-1 relationship with inst using inst as a foreign key. (This relationship information is stored in the RIM specification in the fields foreign key list and foreign key access list.) Transitively, there is a 1-to-N relationship between the shipto entity and the special instruction entity.

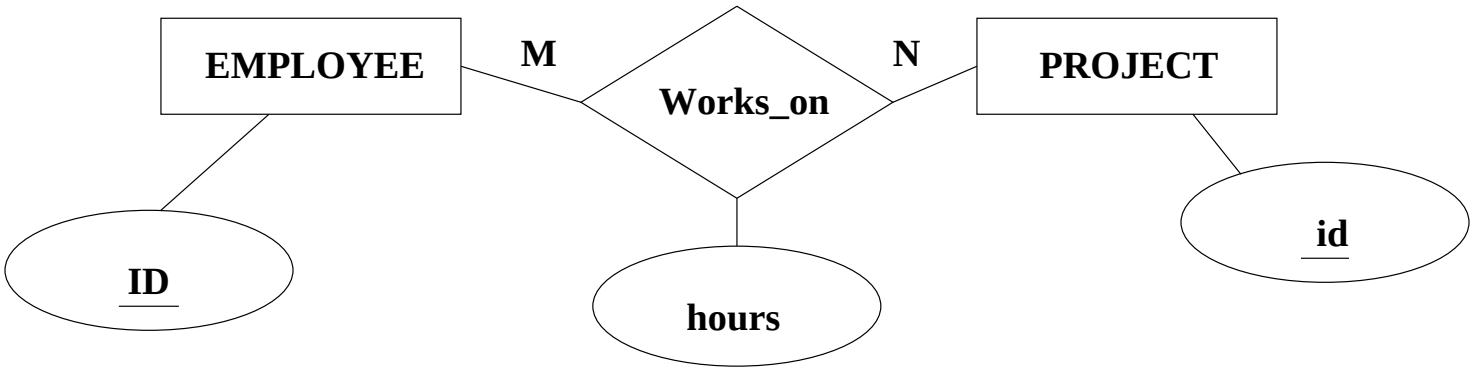

Figure 11: Employee-Project Schema 1

With any 1-to-1 or 1-to-N relationship between entities, the entity with the one participation can effectively add the attributes of the other entity as its own. (In the case of a 1-to-N relationship, it adds multivalued attributes.) Thus, using this relationship information, the shipto entity has key id, and multivalued attributes special instruction codes and special instruction descriptions. This mapping also conforms to the global view.

3.4.3 The Entity-Relationship Conflict

The entity-relationship conflict arises when one schema represents a concept as an entity while another schema represents the concept as a relationship. It is a slightly different problem resolving this conflict depending on whether the global view represents the concept as an entity or as a relationship. The following figures show two example schemas with an entity-relationship conflict. The conflict is over how to represent the fact that an employee may work on several projects.

In Figure 11, an employee is associated with many projects by a M-to-N relationship. This relationship would be represented in a relational model using a works_on relation storing instances of the M-to-N relationship between the entities. In Figure 12, a separate job entity is defined because a job has a unique id. Thus, each time an employee works on a project, that

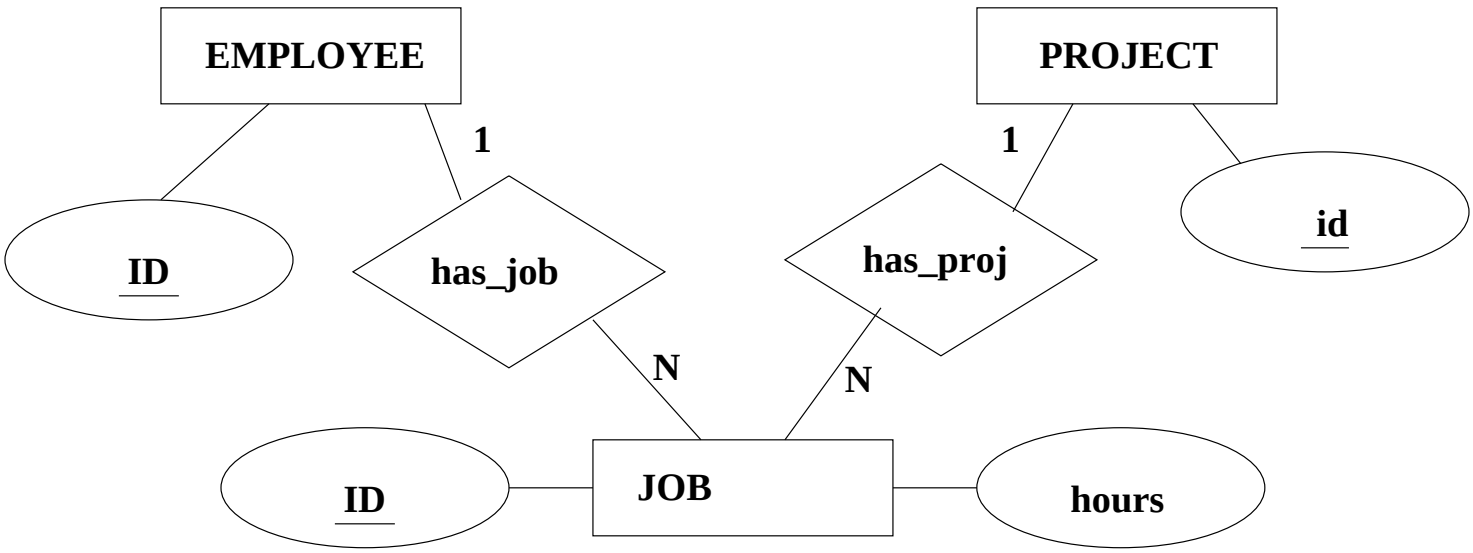

Figure 12: Employee-Project Schema 2

job is assigned a unique id and can be referenced separately from both employee and project. The relationship between employees and projects is accomplished using a 1-to-N relationship between employee and job, and a N-to-1 relationship between job and employee.

Consider the first case where the global schema only has the two entities employee and project and the relationship between them. The first schema is in the proper form, but the second schema is not. Although the employee and project entities are readily identifiable in the second schema, the M-to-N relationship between them is not. However, the relationships has_job and has_proj can be transitively combined to form the desired relationship. The has_job relationship is a 1-to-N relationship between job and employee and can be detected using the foreign key lists. Similarly, the has_proj relationship is a N-to-1 relationship between job and project and can be detected similarly. Transitively, combining these two relationships produces a M-to-N relationship between Employee and Project. As the job entity is the bridge for this relationship, all attributes of the job entity (id, hours) become attributes of the relationship in the global view. Thus, the second schema is integrated into the global view using some simple

foreign key information.

The case where the global schema represents the concept as an entity (Job) instead of a relationship is slightly more difficult because the first schema has no way of defining the job entity. However, the designer responsible for integrating this system could tag the relation works_on with the semantic record name of job. Then, when the first schema is integrated in the global view, the system knows that the record instances in the works_on relation are actually jobs as defined in the global view. Each relationship instance in the works_on relation can then be considered an instance of the job entity with primary key being the combination of the employee id and the project id (the keys of the entities involved in the relation). The relationships between employee and job, and job and project, follow by using the foreign key information provided by the works_on relation.

3.5 Mapping from Relational Schemas to RIM Specifications

The objective of this section is to provide a methodology for specifying relational schemas in RIM. Using schema re-engineering techniques some of the information in the RIM specification can be extracted automatically, while other information requires system designers to manually capture the semantics of the individual systems.

Given a relational schema R with relations $R_1, R_2, ...R_n$, with primary keys $k_1, k_2, ...k_n$, convert R into a RIM specification S. The first step is to automatically extract information from the database at the schema and instance level. The information which can be automatically extracted includes:

- **Table header level** - table name, record size and count, index list, foreign key list and foreign key access list, and possibly record distinction (key) and record duplicate information
- **Table schema level** - field name, field size, field type, and possibly limited categorization

of field use

- **Instance level** - get some of the possible values for categorization fields, and possible relationship cardinalities

Also, for each relation R_i, with primary key attributes K, foreign key attributes F, and other attributes A, it is possible to determine if R represents a relationship or an entity:

- R_i represents a relationship if a subset of its primary key K can be used as a complete foreign key for another relation R_j. Thus, R_i can only be defined by the existence of T which must be an entity.
- R_i represents an entity if no subset of its primary key K can be used as a foreign key for another relation. Thus, the existence of R_i is completely defined by its primary key, which is not derived from other entities.
- An attribute of R_i in A is assumed to be a descriptive attribute of the record instance in the relation. If this is not the case, then the entity or relationship described by the attribute must be named.

The rest of the information in the RIM specification will have to be extracted by hand. Some of this information is very important to the final integration process, especially the renaming of local concepts with the globally standardized names. To add this additional metadata, for each table the schema designer must insert:

- access/sharing mechanisms
- foreign key list
- record update mechanisms
- record type information (does the record represent an entity, relationship, or other?)
- record semantic name - what word or phrase in the global dictionary best represents what this record represents

Also, for each attribute in each table, the designer must assign a semantic name to the field using the global dictionary. It may also be necessary to specify what concept the attribute references. For example, if there are two names in a record, one for a person, and one

for a spouse, each name attribute should include the semantic names person and spouse to differentiate the two name fields. Also, the semantic use of each field must be inserted as this is not easily extracted automatically.

3.6 Combining RIM Specifications and Generating Mappings

Given a set of RIM specifications $S_1, S_2, ..., S_n$ derived from relational schemas, a global dictionary D used to provide common terms for S, the elements of R can be combined and integrated into a global view V by:

- For each entity E in S_i, determine if E is present in some form E' in V by searching for its global name. If E is not present, add E to V as an entity. Otherwise, if E is present:
  - If E' is an attribute, convert E' to an entity and integrate with E.
  - If E' is an entity, match attributes of E with E'. Add attributes of E not present in E'. For each relationship of E, add or integrated attributes to the global view which may be accessible through this relationship.
  - If E' is a relationship, convert E to a relationship and integrate with E. Attributes of E becomes attributes of the relationship E' in the global view.
- For each relationship R in S_i, determine if R is present in some form R' in V by searching for the global names of its corresponding entities. If R is not present, add R to the global view. Otherwise, if R is present:
  - If R' is an attribute, promote R' to a relationship in the global view and integrate with R.
  - If R' is an entity, convert R to an entity and integrate with R'.
  - If R' is a relationship, merge R with R'. Attributes of R becomes attributes of R'. Additional attributes of R can be extracted from the entities participating in the relationship R.

4 Future Work and Conclusions

This work provided an in-depth overview of the schema integration problem by analyzing past solutions to the problem. It categorizes the solutions using a taxonomy and found that current

integration techniques are insufficient because they lacked suitable equivalence comparisons. Finally, it proposed a simple integration language for integrating relational schemas, and argued that the integration language can be used to automatically integrate schemas once the desired information for each component schema has been specified using the RIM specification. Future work includes extending the model to handle more general relational schemas and an in-depth proof of its correctness.

References

[1] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[2] M. Bouzeghoub and Elisabeth Metais. Semantic modeling of object oriented databases. In *Proceedings of the 17th International Conference on Very Large Data Bases*, September 1991.

[3] M.W. Bright, A. R. Hurson, and Simin H. Pakzad. Automated resolution of semantic heterogenity in multidatabases. *ACM Transactions on Database Systems*, 19(2):212–253, June 1994.

[4] M.W. Bright, A.R. Hurson, and Simin H. Pakzad. A taxonomy and current issues in multidatabase systems. *IEEE Computer*, 25(3), March 1992.

[5] Michael L. Brodie. On modelling behavioural semantics of databases. *IEEE*, ??(??), 1981.

[6] O. A. Bukhres, J. Chen, A. K. Elmagarmid, X. Liu, and J. Mullen. InterBase: A multidatabase prototype system. In *ACM SIGMOD*, 1993.

[7] Silvana Castano and Valeria De Antonellis. Semantic dictionary design for database interoperability. In *Proceedings of the IEEE*, 1997.

[8] C. Collet, M. Huhns, and W-M. Shen. Resource integrating using a large knowledge base in carnot. *Computer Magazine of the Computer Group News of the IEEE Computer Group Society*, 24(12), December 1991.

[9] D.A. Cruse. *Lexical Semantics*. Cambridge University Press, 1986.

[10] Suzanne M. Embury, Nicholas J. Fiddian, W. Alex Gray, and Andrew C. Jones. Establishing a knowledge base to assist integration of heterogeneous databases. In *Advances in Databases: 16th British National Conference on Databases*, 1998.

[11] Shashi K. Gadia. Parametric databases: seamless integration of spatial, temporal, belief and ordinary data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 22(1):15–20, March 1993.

[12] Michael Hammer and Dennis McLeod. Database description with SDM: A semantic database model. *ACM Transactions on Database Systems*, 6(3):351–386, September 1981.

[13] Richard Hull. Relative information capacity of simple relational database schemata. *SIAM Journal of Computing*, 15(3), August 1986.

[14] Richard Hull and Roger King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1988.

[15] Richard Hull and Gang Zhou. A framework for supporting data integration using the materialized and virtual approaches. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(2), 1996.

[16] Uwe A. Johnen and Manfred A. Jeusfeld. An executable meta model for re-engineering of database schemas. Technical Report 94-19, Technical University of Aachen (RWTH Aachen), 1994.

[17] W. Kent. The breakdown of the information model in MDBSs. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 20(4):10–15, December 1991.

[18] Won Kim and Jungyun Seo. Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer*, ??(??), December 1991.

[19] R. Krishnamurthy, W. Litwin, and W. Kent. Language features for interoperability of databases with schematic discrepancies. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 20(2):40–49, June 1991.

[20] E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. In *International Conference on Data Engineering*, pages 294–301, Los Alamitos, Ca., USA, April 1993. IEEE Computer Society Press.

[21] E.P. Lim, J. Srivastava, and S. Shekhar. Resolving attribute incompatibility in database integration: An evidential reasoning approach. In Ahmed K. Elmagarmid and Erich Neuhold, editors, *Proceedings of the 10th International Conference on Data Engineering*, pages 154–165, Houston, TX, February 1994. IEEE Computer Society Press.

[22] Witold Litwin, Leo Mark, and Mick Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22(3), September 1990.

[23] J. Meseguer and X. Qian. A logical semantics for object-oriented databases. In *ACM SIGMOD*, 1993.

[24] R. J. Miller, Y. Ioannidis, and R. Ramakrishnan. Schema equivalence in heterogeneous systems: Bridging theory and practice. *Information Systems*, 19(1):3–31, January 1994.

[25] Marek Rusinkiewicz, Amit Sheth, and George Karabatis. Specifying interdatabase dependencies in a multidatabase environment. *IEEE Computer*, ??(??), December 1991.

[26] F. Saltor, M. Castellanos, and M. Garcia-Solaco. On canonical models for federated DBs. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 20(4):44–48, December 1991.

[27] Edward Sciore, Michael Siegel, and Arnon Rosenthal. Using sematnic values to facilitate interoperability among heterogeneous information systems. *ACM Transactions on Database Systems*, 19(2):254–290, June 1994.

[28] A. P. Sheth and G. Karabatis. Multidatabase interdependencies in industry. In *ACM SIGMOD*, 1993.

[29] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogenous and autonomous databases. *ACM Computing Surveys*, 22(3), September 1990.

[30] David W. Shipman. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, March 1981.

[31] Michael Siegel and Stuart E. Madnick. A metadata approach to resolving semantic conflicts. In *Proceedings of the 17th International Conference on Very Large Data Bases*, September 1991.

[32] Chris J. E. Thieme and Arno Siebes. An approach to schema integration based on transformations and behaviour. In *135*, page 30. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, January 31 1994.

[33] D. Weishar and L. Kerschberg. Data knowledge packets as a means for supporting semantic heterogeneity. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 20(4):69–73, December 1991.

[34] Darrell Woelk, Paul Attie, Phil Cannata, Greg Meredith, Amit Sheth, Munindar Singh, and Christine Tomlinson. Task scheduling using intertask dependencies in Carnot. *SIGMOD Record*, 22(2), June 1993.