UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AIM 1011	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Knowledge Base Integration: What Can We Learn from Database Integration Research	5. TYPE OF REPORT & PERIOD COVERED: memorandum
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Jintae Lee	8. CONTRACT OR GRANT NUMBER(s): N00014-86-K-0685, N00014-85-K-0124
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE: January 1988
	13. NUMBER OF PAGES: 33
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research, Information Systems, Arlington, VA 22217	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited

AD-A209 891

DTIC ELECTE JUN 26 1989 S E D

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

database integration,
knowledge base management,
distributed databases,
knowledge base integration,

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper studies the implications of database(DB) integration research for knowledge base (KB)integration. The issues that arise in DB integration and the solutions that have been proposed are examined in order to draw lessons for KB integration. For that purpose, first the difference between DBs and KBs are characterized. The issues are identified that are relevant to both DB and KB integration. Then three existing approaches to DB integration for dealing with these issues are examined. For each of the approaches, I give a brief description, present the solutions offered to the identified issues, and examine their —

Block 20 cont.

adequacy for both DB and KB integration. In conclusion, I discuss the lessons that have been learned from this study.

Massachusetts Institute Of Technology
Artificial Intelligence Laboratory

A.I.Memo No. 1011 January 1988

Knowledge Base Integration: What Can We Learn from Database Integration Research?

Jintae Lee

For
Justification
By
Distribution/
Availability Codes
Dist
Avail and/or Special
A-1

ABSTRACT

This paper studies the implications of database(DB) integration research for knowledge base (KB)integration. The issues that arise in DB integration and the solutions that have been proposed are examined in order to draw lessons for KB integration. For that purpose, first the difference between DBs and KBs are characterized. The issues are identified that are relevant to both DB and KB integration. Then three existing approaches to DB integration for dealing with these issues are examined. For each of the approaches, I give a brief description, present the solutions offered to the identified issues, and examine their adequacy for both DB and KB integration. In conclusion, I discuss the lessons that have been learned from this study.

This report describe research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research has been provided in part by the Office of Naval Research University Research Initiative Program under Office of Naval Research contract N00014-86-K-0685 and in part by the Office of Naval Research under Office of Naval Research contract N00014-85-K-0124.

89 6 23 020

Over the last ten years, researchers have proposed methods for integrating databases. The main goal of this database integration research is a mechanism that allows the user to treat as a single database multiple databases using different data models, query languages, or schemas. There are several motivations for such integration. First, existing databases (DBs) use different representations. Yet we want to make use of them by translating and/or linking them into the representation that we would want for our application. To do so, we need a mechanism for translation and integration. Second, even when we are creating a new DB, we often want to create a distributed DB which is a collection of DBs with their own representations. Such a distributed DB often provides the robustness and the efficiency that a monolithic DB cannot. Third, an integration mechanism would allow the user to avoid dealing with multiple DBs individually. In particular, the user need not formulate multiple queries or learn different database languages.

Similar motivations exist for integrating knowledge bases. Knowledge acquistion is a costly process; so there is a strong motivation for reusing existing knowledge bases (KBs). Yet even when there are KBs that contain the desired knowledge, we find it difficult to use them. Partly, this is because in most knowledge-based systems there has not been a clear boundary between a knowledge base and other components, such as an inference mechanism. The recent suggestion that we take a functional view of the knowledge base [Brachman and Levesque 86] does much to address this problem. But even when we have well-defined knowledge bases, it is not easy to integrate them because of the differences in the languages and the ontologies they presuppose. For example, if we have one knowledge base with the knowledge *All STUDENTs are under stress* and another KB with the knowledge that *Jin is a STUDENT*, we might not be able to conclude that *Jin is under stress* because, among other things, STUDENT in the KBs might not refer to the same entity. (For the purpose of this paper, I use the term 'object' to refer to anything like an entity, a relation, or an attribute that we want to refer to as a unit while reserving the term 'entity' for those things that we want to distinguish from attributes or relations.)

The purpose of this paper is to study the implications of the DB integration research for KB integration. In this paper, I examine the issues and the solutions that have been studied in DB integration research and try to draw lessons from them for KB integration. In the first section, I present an example which concretely illustrates the goals of DB as well as KB integration, and which will be used to illustrate the points in the rest of the paper. In the next section, I analyze the differences between DB and KB so that we can determine the relevance of DB integration research for KB integration. In the third section, I present what I believe are the issues that are important for both DB and KB integration. In the fourth section, I examine the three approaches to DB integration, which are respectively represented by the systems discussed in the three assigned papers. For each of the approaches, I briefly describe the approach, study the solutions it offers to the identified issues, and examine their adequacy both for DB and for KB integration. Finally, I conclude the paper by

discussing the lessons that I believe this analysis has produced and drawing a picture of a KB integration mechanism based on these lessons.

1. Examples

In this section, I present an example of DB as well as KB integration to provide concreteness for the discussions that follow. For instance, this example will be used to illustrate the differences between DB and KB integration as well as the various constructs that the different approaches to DB integration offer. I start with an example of DB integration.

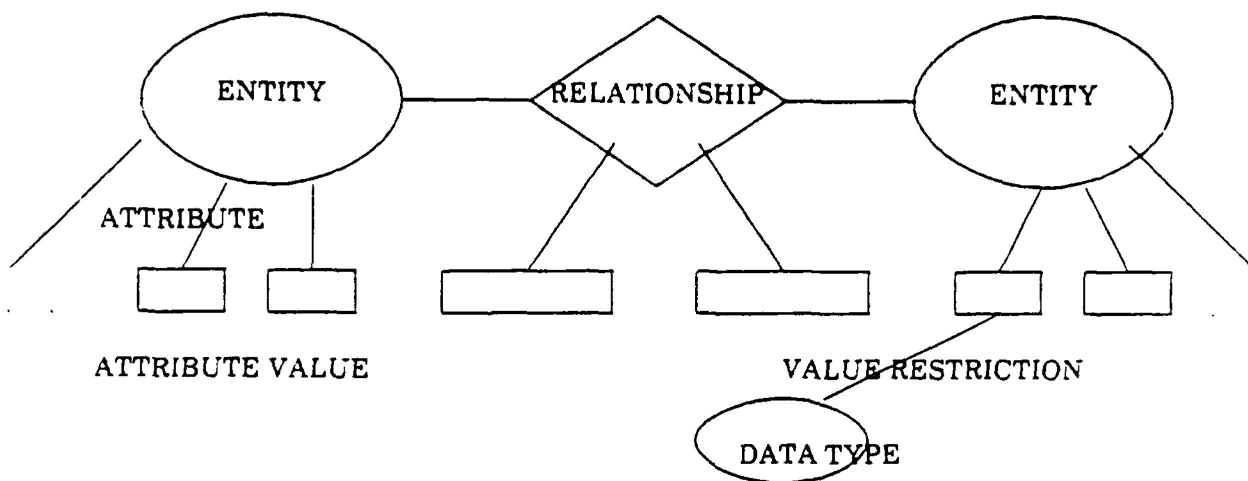
Suppose we have two databases, DB1 and DB2. Fig. 1a shows the entity-relationship data model that we will assume both DB1 and DB2 use. A data model is a description of generic categories that are presupposed by a database. All the operations for a given data model are defined with respect to the objects that appear in the model. Fig. 1b shows partial schemas for DB1 and DB2. If we regard a data model as defining a grammar, then a schema defines the vocabulary. Fig. 1c shows relational table representation of some of the entities in the schemas. Fig. 1d shows a query that we might want to ask of these DBs. If the tables from Fig. 1c were in fact from a single DB, then a natural thing to do to answer the query would be to join across tables. We would first select from the AREA-EXAM relation all the area exams scheduled for this semester, then select from the GRAD-STUDENT relation the graduate students from EECS, and finally join the two resulting tables to get information about the area exams given this semester for EECS graduate students.

However, when the tables belong to different databases, this joining may not work for several reasons. The data models of the databases might be different: one might use a relational model and the other a functional model. Even within the same data model, the particular languages they use might be different--say, SQL and QUEL. Even when the same language is used, the data schemas for the different databases are usually different. StudentID in DB1 may not be the same as StudentID in DB2. In Section 3, I identify and categorize the differences that need to be resolved to establish equivalence across different databases. A main goal of DB integration research is to provide a mechanism for resolving these differences so that the user can treat multiple databases as one despite these differences.

The goal of KB integration is similar. Fig. 2a shows a 'knowledge model' asumed by the knowledge representation language, KANDOR. Fig. 2b shows the vocabulary defined defined in that model. Appendix I shows the actual KANDOR code representation of this vocabulary. KB1 contains the knowledge about area exams. KB2 contains the knowledge about departments. As we see from Figure 2, there is much more machinery in a KB than in a DB. In the next section, I discuss what more a KB offers with this additional machinery than a DB .

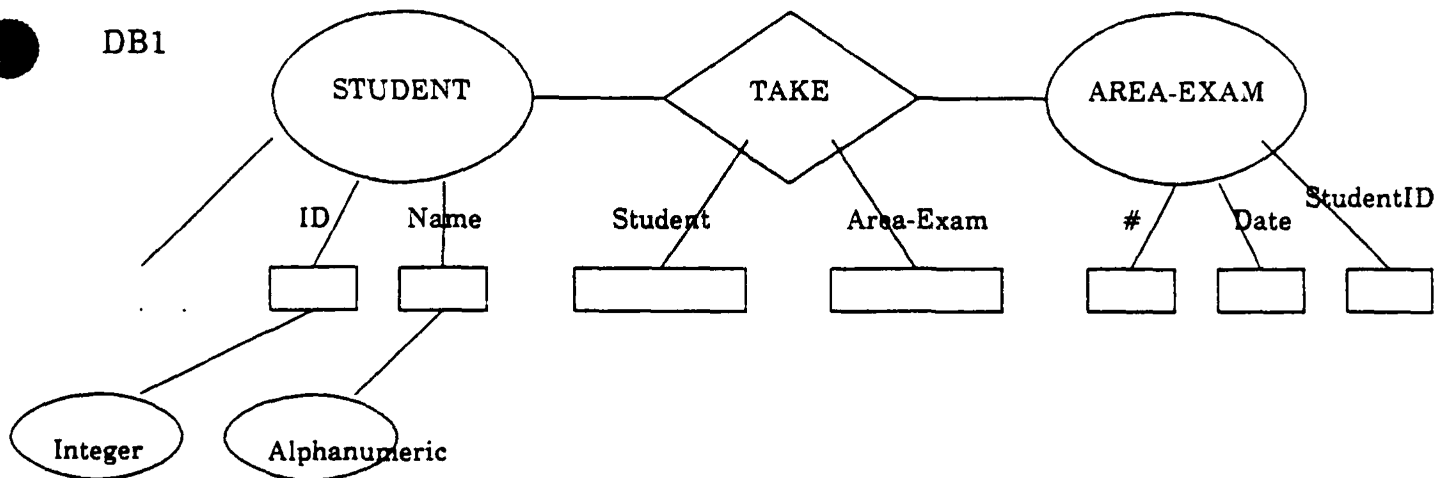
FIGURE 1. SOME EXAMPLES OF REPRESENTATION IN DATABASES

(a) ENTITY-RELATIONSHIP DATA MODEL



(b) DATABASE SCHEMAS

DB1



DB2

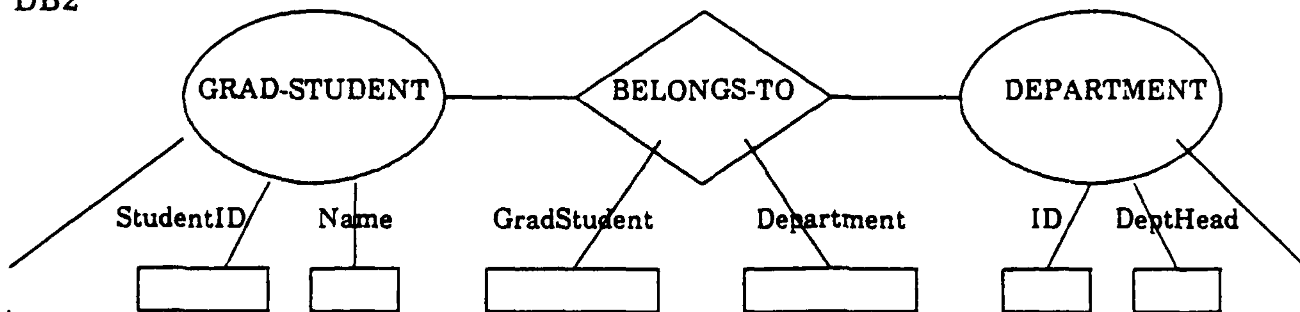


FIGURE 1. SOME EXAMPLES OF REPRESENTATION IN DATABASES

(CONTINUED)

(c) RELATIONAL TABLES

AREA-EXAM IN DB1

#	Date	Topic	. .
17	12/16/87	DB/KB	
18	12/22/87	CSCW	
.	.	.	

GRAD-STUDENT IN DB2

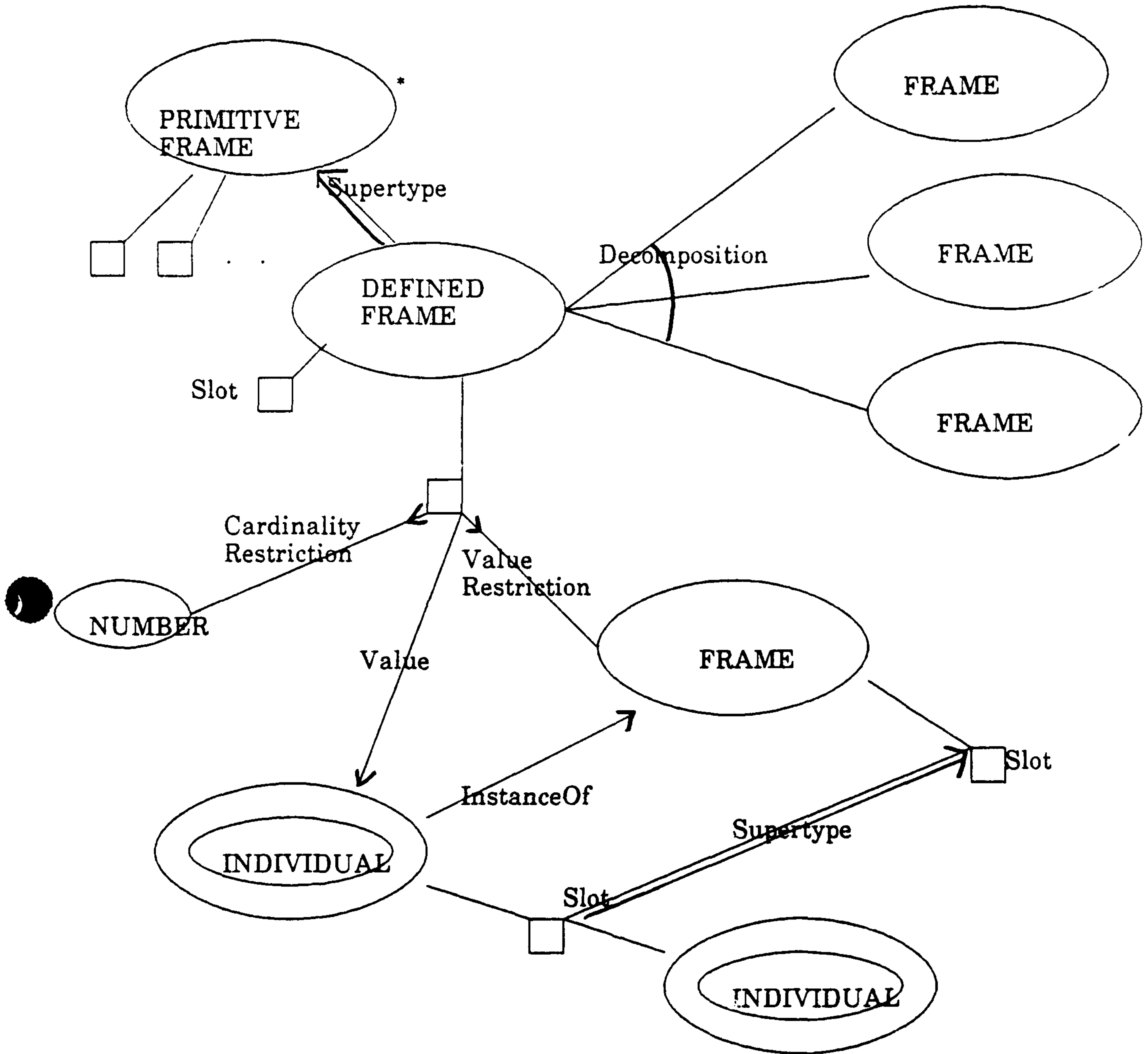
StudentId	Name	Support	. .
154-52-5033	JIN	RA	
123-45-6879	DAR	RA	
.	.	.	

(d) A QUERY

GIVE ME ALL THE INFORMATION ABOUT THE AREA EXAMS GIVEN THIS SEMESTER FOR GRADUATE STUDENTS FROM EECS DEPARTMENT.

FIGURE 2. SOME EXAMPLES OF REPRESENTATION IN KNOWLEDGE BASE

(a) 'KNOWLEDGE MODEL' FOR KANDOR



The 'model' above is only for illustration purpose. To really see the formal model, see [Patel-Schneider 84].

FIGURE 2. SOME EXAMPLES OF REPRESENTATION IN KNOWLEDGE BASE

(b) KANDOR 'KNOWEDGE BASE SCHEMAS' (CONTINUED)

KB1

EXAM
Identifier
AREA-EXAM
PERSON
ID
Date
l
ID
Date
Student
Name
STUDENT
JIN
Address
ID
Name
Address

KB2

PERSON
GRAD-STUDENT
Name
Interest
ID#
Name
Dept
JIN
DEPARTMENT
Dept
ID#
EECS
Name
Office
Name
Office

What is a knowledge base? I take the functional view of a knowledge base by treating a KB as a black box which provides answers to a set of well-defined query operations. The set of KB-query functions for Kandor is shown in Appendix II. [Brachman et. al. 83] points out and persuasively argues for the advantages of taking the functional view of KB: a better semantic model free of implementation biases, a basis for comparing knowledge bases in terms of the knowledge that can be represented, and so on. For our purposes, one big advantage of this view is that a knowledge base becomes modular and transportable. That is, taking this view allows us to isolate a knowledge base from other components of a knowledge-based system so that we can treat it as a unit that can be reused or integrated with other such units.

In this paper, I also assume that the representation language used by different KBs is the same, and we will use KANDOR as the language for illustration. Most DB integration researchers also make the assumption that different DBs use the same data model. When they do not, they assume that different data models can be translated into a single data model. Only then they address the integration issues within that model. Hence DB integration research does not address the problems in translating between different data models. I do not either in this paper. Given these assumptions, the goal of KB integration then becomes providing a mechanism by which the user can use these query functions without worrying about the differences between the knowledge bases. For example, the meaning of STUDENT in KB1 and KB2 might not be exactly the same. We want to be able to establish an appropriate relation between them; and once the relation is established, we want the system to do the right thing for each KB involved when we make a query about STUDENT.

2. How is a KB different from a DB?

Since the goal of this paper is to examine the implications of DB integration research for KB integration, it is important to be clear about the differences between DB and KB. These differences will help us determine which part of DB integration research is relevant and point to additional issues that are not addressed by that research.

One way to characterize the differences between a DB and a KB is to compare them at the symbol level [Newell 81]. That is, we can view both DBs and KBs as data-structuring packages and identify what constructs one offers that the other does not. For example, most KBs have an inheritance mechanism while most DBs have none. However, we could also compare them at the knowledge level [Newell 81] in terms of the kinds of knowledge that they can represent. As [Brachman and Levesque 86] argue, a database can be viewed as a limited form of a knowledge base. The kinds of knowledge a DB can represent form a subset of the kinds of knowledge that a more full-fledged KB can (eg. a KB written in KANDOR). In particular, a database is not able to express certain incomplete knowledge such as disjunction (The ID of JIn is either 154525033 or 888123456) or existential quantification (The ID of Jin exists but is unknown). DBs limit their expressive power this way to gain computational tractability.

If we take the functional view of a knowledge base, this knowledge level view of DB, i.e. the view of DB as a limited form of KB, is a natural one to adopt. A DB is a knowledge base in the sense that it can be treated as a black box that can provide answers to a certain set of queries, but the kinds of queries it can answer are more limited than other KBs because of its limited expressive power. (For the sake of convenience, however, I will use the term 'knowledge base' to refer to KBs other than DBs in this paper without thereby implying that a DB is not a KB.) Given this view of the relationship between DB and KB, the differences between them that matter for the purpose of KB integration are the differences that their query languages reflect. (We also need to look at their update languages if we are interested in integrated updates. But in this paper, we assume, as most DB integration researchers do, that updates are done locally and we are only interested in integrated retrieval of knowledge).

Examining the set of query functions for KANDOR (Cf. Appendix II) reveals some of the representative differences between a DB and a KB. Fig. 3 lists the features present in KANDOR but not in most DBs. Some of these features such as the distinction between primitive and defined objects are fairly unique to KANDOR and its relatives (KLONE, NIKL), but most of the features are present in other KBs. In the next section, I discuss some of the issues that these additional features raise for KB integration.

FIGURE 3. Features Present in KANDOR but Missing in DBs

Abstraction relation among objects:

generalization *e.g. (knd-super 'STUDENT) -> PERSON*

decomposition *e.g. (knd-decompositions 'PEOPLE) -> (MEN WOMEN)*

Finer distinction among objects:

primitive vs. defined *e.g. (knd-primtive? 'TRIANGLE) -> NIL*

class vs. individual *e.g. (knd-types 'JIN) -> (STUDENT PEOPLE-UNDER-STRESS)*

More general value restrictions:

non-atomic objects *e.g. (knd-value-restrictions 'JIN 'DEPT) -> DEPARTMENT*

(DEPARTMENT above is not a string but a class of certain objects.

cardinality restriction *e.g. (knd-min-restriction 'JIN 'PARENTS) -> 2*

More general attribute values:

non-atomic objects *e.g. (knd-slot-value 'JIN 'DEPT) -> EECS*

(EECS above is an object not a string.

3. Issues

In this section, I identify and categorize the issues that arise in DB integration. I indicate to what extent these are also issues for KB integration. First, let us look at what are all possible ways that two DBs can represent the same piece of information-- concept, data, etc. From the functional point of view, it means looking at all the different kinds of objects that a given query language allows and determining what kinds of differences in representation these kinds of objects allow. To be more specific, if we look at the entity-relationship data model in Fig. 1a, the set of distinctions allowed in the model consists of: entity, relationship, attribute, attribute value, and value restriction. From this fact, we can gather that the same concept might be represented as an entity in one DB but as a relationship in another DB or as an attribute in yet anther DB. Also, different DBs might associate different value restrictions for a given attribute. Fig. 4 lists these possible differences.

FIGURE 4. Representation Differences To Be Resolved

Name Differences
synonym :*Two entities are the same but have different names.*
homonym :*Two entities are different but have the same name.*
Granularity Differences
:*an entity is a subtype of another*
Value Restriction (VR) Differences
;*Two attributes judged to be equivalent have different value types.*
Structural Differences
:*Two entities judged to be equivalent have different structures: eg. entity vs. attribute.*
Ontological Differences
:*The entities result from different partitions or categorizations of the domain.*

There may be name differences. Two entities or attributes in different DBs might represent the same object but have different names (synonyms): e.g. QUALIFYING-EXAM and GENERAL-EXAM. On the other hand, two entities in different DBs might have the same name but really represent different objects (homonyms). For example, STUDENT in one DB may mean only full-time students whereas STUDENT in another DB may include any person taking a course. Then there may be value restriction differences. For example, one DB might require Alphanumeric for the attribute ID (EECS) whereas another DB might require Integer (6 as in 'Course 6'). The same concept may be represented as different kinds of objects (structural differences). For example, the concept of department may be represented as an entity in one DB but as an attribute in another.

Granularity differences exist when different DBs represent entities at different levels of abstraction. For example, one DB may have STUDENT, but another may have only

GRAD-STUDENT. We want to establish a subtype relation between them so that when a query is made about STUDENT, the information in all of its subtypes, including GRAD-STUDENT, can also be retrieved. The relational model shown in Fig. 4a does not show the possibility of this type of difference because it cannot express such a relation. However, many of the data models studied in DB integration research, such as the functional data model and extended entity relationship model, do allow the expression of the subtype relation. In those models, these differences need to be resolved in some way.

Ontological differences are more fundamental. They arise from the differences in the way that different DBs partition or categorize their domain. For example, one DB may categorize all people on the basis of their position (FACULTY, STUDENT, STAFF, etc.), but another DB might categorize them on the basis of their income (HIGH-, MIDDLE-, LOW-INCOME). Although ontological differences are common, most DB integration research avoids dealing with these differences by assuming that the judgement of how to map entities in such cases is externally supplied-- either by people or by another procedure. Hence, I do not discuss resolving these differences in this paper.

Examining Fig. 2a shows that the same kind of representational differences need to be resolved for KB integration as well. Name differences may certainly exist; so may granularity differences due to the existence of supertype relations (Supertype, InstanceOf). Value restriction differences also exist, and resolving these differences among KBs may require more complex solutions than among DBs because in KBs the value restrictions are defined not in terms of simply data types like Integers but any arbitrary entities (e.g. DEPARTMENT). Another kind of value restriction-- cardinality restriction-- also has to be dealt with. There are more kinds of structural differences as well. Due to the finer distinctions among the entities, the same concept may be represented as a class or as an individual or as a slot, and as a primitive or as a defined object. There are other kinds of differences that need to be resolved. Two KBs might have different decompositions of a given class. Also, some KB languages like SRL have special constructs for building an object whose parts are other objects. In such cases, two KBs might have different parts for an object that one wants to treat as equivalent.

I do not propose any solution for resolving these differences in this paper. The goal of the paper is mainly to examine the adequacy of the solutions from DB integration research for KB integration. I mention them only to indicate that a KB integration mechanism has to provide some additional ways to resolve them. Also, as we will see, some of these differences also require extensions to the DB solutions if they are to be used for KB integration even for those differences present in both DB ad KB.

(To be sure, different query languages have different data models. For example, relational query languages such as SQL do not make a distinction between entities and relationship. On the other hand, functional query languages such as DAPLEX presupposes the functional data model, where one can express a subtype relation that one cannot in the relational data model. Likewise, there is no single knowledge model for KBs. Different languages offer different sets of objects. When we talk about DB or KB integration, either we are presupposing a common model shared by all the

components, or different models that themselves will have to be integrated. As mentioned above, DB integration research does not have much to say about integrating different models. In the following discussion, I assume that the integration is done among DBs or KBs using the same model. In particular, I use the entity-relationship model (and occasionally the functional data model) for DB integration and the model of KANDOR for KB integration to illustrate the discussions in this paper.)

Fig. 5 lists other important issues that need to be considered in DB as well in KB integration, once we have a mechanism for resolving the differences discussed above. First, there is a set of issues about how well such a mechanism supports evolution. For example, given that the databases or their schemas inevitably change over time, can the mechanism accommodate these changes gracefully? In particular, how much consensus is required among the parties involved (eg. database administrators) to introduce changes, say in the definitions? Once a change is made, what sort of propagation and updates does it require to reflect the changes across the databases involved? Another set of questions addresses the cost to set up and maintain the mechanism. Different approaches in DB integration present different solutions to these issues with different tradeoffs. In the next section, we will examine how adequate these solutions are in the context of both database and knowledge base integration.

FIGURE 5. Other Issues in DB/KB Integration

Evolution:

Need for Consensus

;How much agreement is necessary to introduce a change?

Need for Updates

;Once a change is made, what needs to get updated and to what extent?

Costs:

Setup Cost

Maintenace Cost

4. Database Integration Research

The three papers assigned represent three different approaches to database integration. Multibase [Dayal & Hwang 84] represents a global schema approach, where the local schemas for different databases are merged into a global schema, against which the user makes his queries. Federated Architecture [Heimbigner & McLeod 85] represents a locally global schema approach, where there is no single global schema but each database imports objects from other databases and then constructs its own global schema that integrates them with its own schema. MRDSM [Litwin & Abdellatif 86] represents a multidatabase language approach, where there is not even a locally global schema but only a language specially designed to provide a uniform interface to multiple databases. Each of these approaches offers different solutions to the issues discussed in the last section with

different tradeoffs. For each of the approaches, I first give a brief exposition, discuss the constructs it offers to resolve the semantics issues, and examine its adequacy in the context of both DB and KB integration.

4.1 Global Schema Approach

In this approach, local schemas are integrated into a global schema and the queries that the user makes are defined in terms of the entities and the attributes in the global schema. A query expressed in terms of a global schema is then translated to local schemas by using the mapping between the entities in the global schema and those in the local schemas.

This approach is one taken by the most people in DB integration research. The differences among them lie in the assumptions they make about the data models (whether a specific data model is required for all DBs or not), about the kinds of input (views, local schemas, other specifications), about the kinds of output (global view, global schema, set of conflicts, etc.), and the kinds of strategies they take (Cf. Batini et.al. 86). Although I use Multibase as an instance of this approach and use the constructs it offers to illustrate how the representation differences are resolved, the main points I make are applicable to the global schema approach in general.

Representation Differences

The constructs that Multibase offers are: virtual entities, generalization, procedural attachment, and auxiliary DB. I will explain them as I discuss how they are used to resolve each of the semantic issues.

Name Differences

Name differences are easy to handle. If two entities in different databases are judged to be the same with only name differences (synonym), then they are represented as one entity with a single name in the global schema. The map between the local names and the global name is maintained so that a query involving a global name can be translated to local queries appropriately. On the other hand, if two entities in different databases have the same name but are judged to be different (homonym), then they are renamed so that they appear in the global schema as two distinct entities. If we assume that the judgement of when two entities are equivalent or different comes from outside, as we did above, then renaming will also work for KB integration. However, there is a difference between DB integration and KB integration in the process of identifying such equivalences.

Whatever process was used, whether by machine or by human, to decide on equivalences among entities for DB integration, we can assume that the process is going to be more complicated in KB

integration by virtue of the fact that attribute values are themselves objects in KB. For example. take a simplistic criterion that says, "Two entities, X and Y, are the same if for each A of X, there is an attribute of Y that is equivalent to A" Let's say that two attributes are the same if their 'meanings' are judged to be the same by the user and they have the same domain type. (In this paper, the term 'user' is used in two different ways: the end user and the database designer or administrator. The contexts should make it clear in which of the senses the term is used.) For example, the attribute ID and StudentID may be judged to be the same and they have the same domain type, say Alphanumeric. In this case, at least the equivalence of entities has been reduced to the equivalence of attributes and their data type. However, since in a KB the domain type for an attribute is not a data type like Integer or String but an arbitrary object, deciding whether two domain types are the same requires deciding whether the two objects themselves are equivalent, which might in turn require determing the equivalence of their attributes and the domain types, and so on. A KB integration mechanism may not automatically deduce these equivalences for the user, but it should be able to provide consistency checking for the equivalences that may be externally provided.

Granularity Differences

Suppose the user decides that GRAD-STUDENT in DB2 is a subtype of STUDENT in DB1. Then in the global schema, both GRAD-STUDENT and STUDENT appear but the former as a subtype of the latter. Note that, to do so, the data model for the global schema has to be expressive enough to represent this generalization relationship although each of the local schemas may not be. Multibase uses the functional data model, which uses the supertype construct to express such a relationship. Given this relationship, when a query is made about STUDENT, the query is then translated not only into a query about STUDENT in DB1 but also into queries about all of its subtypes-- in particular, about GRAD-STUDENT in DB2. This is useful because some of the data in the GRAD-STUDENT table of DB2 might not be present in the STUDENT table of DB1. KB representation languages also can express supertype relations. So these differences can be resolved in the sme way.

Value Restriction Differences

Value restriction differences arise when different databases associate different domain types with the attributes that the user wants to treat as equivalent. For example, Department might have the attribute restriction Alphanumeric (EECS) or Integer (6 as in 'Course 6'). To compare or integrate such values, Multibase allows the user to create an auxilary DB which contains a table relating the two types of values. In a KB, the use of an auxilary DB would naturally correspond to the use of an auxilary KB that contains the relevant translation knowledge. Since a KB is more general than a DB, we might expect that a more general sort of translation is possible by using an auxilary KB. In both DB and KB, the adequacy of the solution based on auxiliary KB depends on the other constructs we have discussed because now the auxilary KB is another KB that needs to be integrated with the others. [Dayal & Hwang 84] mentions that one can associate a formula or even a general procedure for translating from one to another type of values. Since the paper does not discuss in detail how that is

done, I cannot discuss the adequacy of this solution. But in general, we should make sure, especially in KB, that the use of procedures does not distort the semantics of the knowledge bases involved.

Structural Differences

Structural differences arise when a concept appears in different structural forms (eg. entity vs. attributes) in different databases. For example, Department might be an attribute in one DB but an entity in another. Fig. 6a. illustrates one example. The user, i.e. the database designer in this case, can resolve these differences by defining virtual entities or virtual attributes that Multibase allows. That is, the user can define an entity in the global schema and relate it to the entities in the local schemas. For example, Fig. 6b. illustrates the solution for our example. DEPARTMENT1 is a virtual entity and BELONGS-TO1 is a virtual relation because there is really no DEPARTMENT entity or BELONGS-TO1 relation but only an attribute called DEPARTMENT in DB1. These virtual entities or relations have been created so that the user can create a single set of entities or relations as supertypes of the corresponding components from the two DBs.

It seems that any integration mechanism, whether for a DB or for a KB, should allow some constructs for defining virtual entities or attributes to resolve such structural differences. The question is rather what kinds of structural differences exist and what specific constructs we need to resolve them. Although Multibase provides a general construct for defining a virtual entity, the paper does not discuss any specific operators available to do so. For example, the syntax for defining a virtual entity is ***DEFINE ENTITY TYPE <entity type> (<target-list>) WHERE <Qualification>***; but it is not clear what operators are avialable for specifying Qualification. On the other hand, the federated architecture approach discussed in the next section is quite specific about the operators available for defining a virtual entity. So I leave further discussion on virtual entities for the next section.

Evolution

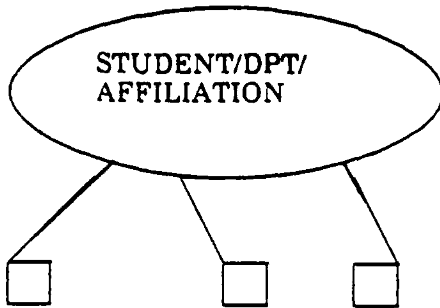
Need for Consensus

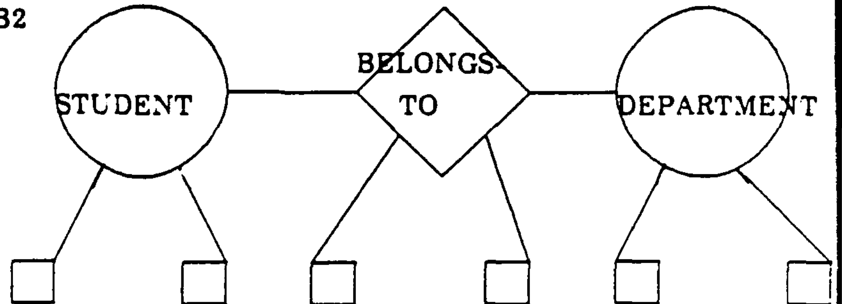
Any attempt at integration requires agreements on which of the objects are equivalent, or in general how they are related. But the global schema approach requires more of such agreements because it has to build a global schema that integrates the entirety of all the local schemas. For database schemas with any realistic size, obtaining consensus on how the different entities should be integrated becomes virtually impossible because of the coordination cost involved as well as disagreements. Furthermore, whenever we need to change a global schema, we also need consensus because changing a global schema will affect all the local databases.

Need for Updates

FIGURE 6. STRUCTURAL DIFFERENCE RESOLUTION IN MULTIBASE

LOCAL SCHEMAS





(b) AN INTEGRATED SCHEMA

STUDENT

BELONGS-
TO

DEPARTMENT

STUDENT1

BELONGS-
TO1

DEPARTMENT1

STUDENT2

BELONGS-
TO2

DEPARTMENT2

When a change is made to a local schema, the change has to be incorporated into the global schema. Incorporating such a change is likely to be non-trivial if it disturbs the equivalences upon which the global schema is based. Since there is no modularity nor dependency record in the way that the global schema is designed, such a change may imply a chain of modification or an entire reconstruction of the global schema. Introducing a change into a global schema has graver consequences. It may require changing the mappings between the global schema and all the local schemas that have been integrated.

Costs

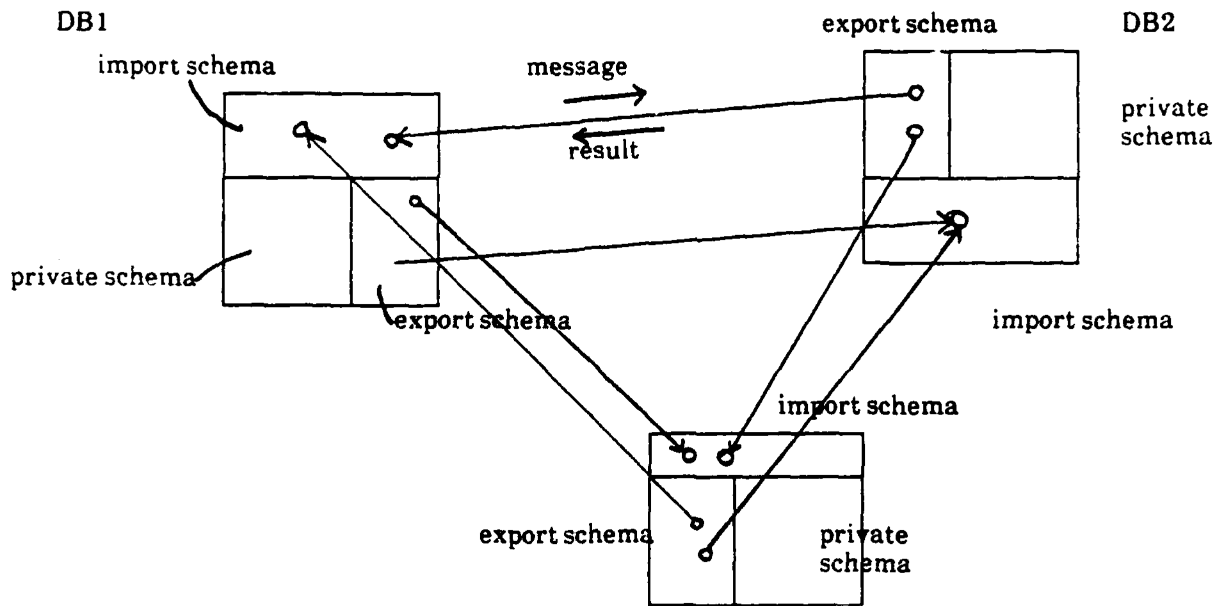
The initial cost of setting up a global schema is large in both efforts and the domain expertise involved. Even when one manages to build a global schema, it is likely to contain so many entities and attributes that it tends to be incomprehensible even to database managers. The maintenance cost includes translating a global query into local queries. That is a cost that any integration approach probably would have to bear, but the maintenance cost also includes the cost involved in supporting its evolution. As we have seen, obtaining consensus and updating changes to support evolution is very costly in this approach. For these reasons, the global schema approach is unlikely to work except in cases where the local database schemas are small and do not change often. However, it does not mean that the constructs we have discussed for resolving the representation differences are of no use or unlikely to work, as we will see.

4.2 Locally Global Schema Approach

In this approach, there is no global schema. Instead, each database imports objects from other databases and creates its own global schema by integrating them into the objects of its own. The Federated Architecture proposed by Heimbigner and McLeod represents an instance of this approach. Fig. 7 shows the structure of the proposed architecture. A database joins a federation first by importing a federation dictionary which contains the federation-wide information necessary for minimal coordination. The database then declares an export schema, which describes the set of objects it is willing to share with others, and imports objects from the external schemas of other databases. As it imports objects, it integrates them with the existing schema, which itself is a result of integrating its own schema and the previously imported objects. A query is made using the objects in this locally global schema; when a reference is made to an object, a message of an appropriate type is sent to the database from which the object was imported. Appropriate message types are also imported when the objects are, so that if a proper message type is used to request information about external objects, then the original database which exported those objects knows how to process those messages. The results

are then sent back to the requesting database, which integrates them with the results from other databases or from its own internal query.

FIGURE 7. FEDERATED ARCHITECTURE



Before discussing the specific solutions this approach provides to resolve the representation differences, it is worthwhile to be clear about the similarities and the differences between this approach and the global schema approach. Both have to build a global schema in the sense that they need to integrate the objects from other databases with their own. Hence, all the issues and the solutions discussed in the previous section on Global Schema Approach are relevant to the present approach as well. A difference is, however, that the global schema in this approach is much easier to build because each database can build its own, based on the set of mappings among objects that are useful for its own purpose, i.e. for the set of applications that the database is used. The global schema is also easier to build because the global schema need not integrate the entirety of local schemas but only one's own schema with those objects that one imports. Another difference is that the global schema construction is an incremental process managed by the local schema who owns it, rather than a one-time process that involves all the local schemas. Yet another difference is that a change in local schema does not have to be propagated unless it is a change in its external schema. Also unlike in the global schema approach, a user of a given DB would continue to use the same query language to access multiple DBs.

Representation Differences

The constructs offered by Heimbigner and McLeod for resolving the representation differences are:
Object equality function

Type Derivation Operators: concatenate, substraction, cross-product, subtype
Map Derivation Operators: composition, inversion, extension, restriction, cross product, discrimination, projection, selection

[Heimbigner & McLeod 85] provides a description for each of these constructs (pp. 262-265), which I will not repeat here except for those used for illustration below. The discussion below will be brief because the constructs used to resolve the representation differences in this approach are essentially the same as those in the global schema approach. Although there are many more operators available for relating entities and maps, they are all used to define virtual entities or attributes. We can regard them as more detailed specification of the language that can be used for defining Qualification clauses in virtual entity definition in Multibase. So the constructs discussed below do not offer any new solutions for resolving the differences but only serve to illustrate the virtual entity solution of Multibase more concretely.

Name Differences

When a database DB1 imports a type, say AREA-EXAM, from another database DB2, it can rename the type to, say, QUALIFYING-EXAM. DB1 associates with the new type QUALIFYING-EXAM a derivation expression "AREA-EXAM > DB2" so that it will know that the new type is a derivation of the type AREA-EXAM of DB2. Homonyms can also be renamed accordingly.

Granularity Differences

One of the type derivation operators, Subtype, allows the user to create a subtype of another type based on the values of an attribute of the latter. So if DB1 wants to establish a subtype relation between STUDENT in its own schema and GRAD-STUDENT from DB2, then DB1 first imports GRAD-STUDENT and then define it as a Subtype derivation of STUDENT.

Value Restriction Differences

Differences in domain types can be resolved via object equality functions, which are arbitrary host procedures that translate a certain type of value to another type of value. For example, if two DBs represent the same department respectively as EECS and as 6, then an object equality function is written to translate EECS to 6 so that the two values can be compared.

Structural Differences

If DB1 wants to import an attribute Department from DB2 and treat it as an entity, then DB1 would create a type called Department and define, by using the map derivation operators, one of its attributes to be a derived attribute of the attribute Department of DB2.

The other operators that we have not mentioned are used to resolve other structural differences that we have not explicitly mentioned. For example, a type derivation operator Cross Product takes as arguments n types (say, Month, Day, Year) and creates a new type (Date) of n-tuple, each element of which comes from each of the n types. An example of a map derivation operator is Composition, which take two maps (attributes like Manufacturer of type Airplane and Name of Manufacturer) as arguments and defines a new map (the attribute, The Name of Manufacturer of Airplane) as a composition of the arguments. These operators, as mentioned, can be regarded as further specification of the language for defining virtual entities.

Evolution

Need for Consensus

Consensus is required on how to integrate external types into the locally global schema. But, as mentioned above, this consensus need to be obtained only locally within the database. The same is true with the consensus required for any change in the global schema; only local consensus is needed.

Need for Updates

Only those changes made to objects in one's external schema need to be propagated to other DBs. In fact, Heimbigner & McLeod mentions an implicit contract among the databases in the federation. In this contract, "[a member DB] guarantees that it will not modify the definition (structure or semantics) of the exported elements unless it notifies the importer. By this contract, [the importers] also agree to notify the exporter when it no longer requires access to the element." When a change is made to an entity in the external schema, its importers "have the option of either relinquishing access to the element, or notifying the exporter that the modified element is an acceptable replacement for the original element" with possible readjustment in its global schema. Still the readjustment that has to be made in such cases can be subtantial because, as in the global schema approach, a change to a map can trigger changes in other maps, upon which the global schema is based.

Costs

The setup cost consists of creating a common dictionary that is to be shared by all the members in the federation, creating an export schema, importing external entities, and constructing a global schema that integrates imported entities with one's own schema. The cost is still substantial, although as mentioned above the global schema can be integrated incrementally. The maintenance cost involves identifying external entities, translating them into the original entities they were

derived from, sending messages requesting information about these entities to appropriate databases. and translating a message to database queries as well as the costs for supporting evoution. So the maintenance cost is fairly substantial, but the benefit one gets for this cost is the local autonomy of the component databases.

4.3 MultiDB Language Approach

In this approach, no global schema is constructed. Instead, there is a multi-database query language that allows the user to formulate a query in a generic way, which then gets translated into different queries for different databases. The expansion of a generic query into multiple queries is not unique to this approach; it also takes place in the global schema approach. The difference is, however, that in the global schema approach, the query is made against the global schema. As such, the query language used in that approach is no different from any other query language that is used for a single database. However, a multi-database query language like MDSL [Litwin & Abdellatif 86] is specifically designed to express generic queries without the need for a global schema. MRDSM (Multics Relational Data Store Multidatabase) is a multidatabase system that uses MDSL to allow generic access to multiple databases.

Representation Differences

The constructs MRDSM provides are the following:
Abbreviations:
Multiple Query:
  multiple identifier,
  semantic variable (implicit, explicit domains),
  options (mandatory, optional, one-of)
Incomplete Queries
Dynamic Attributes

Name Differences

Homonyms are dealt with by prefixing the DB names to the entity or attribute names. For example, if we want to distinguish STUDENT in DB1 from STUDENT in DB2, the user simply refers to them as DB1.STUDENT and DB2.STUDENT. The user can deal with synonyms in several ways, depending on how many details he knows about the databases. If he knows the exact names of the entities or the attributes, he can use a *semantic variable* with an explicit domain. A semantic variable is a variable that ranges over attribute names, and a semantic variable with an explicit domain is one where its domain is defined explicitly by enumerating its members. For example, if DEPARTMENT in one DB and DEPT in another DB are judged to be equivalent, then the user declares a semantic

variable DPT whose explicit domain consists of {DEPARTMENT, DEPT} and formulates a query using these variables (*Select DPT where DPT.Name = EECS*). The multiDB language processor will then determine which identifiers belong to which DB and translate the query into queries appropriate to the databases involved (i.e. DEPARTMENT.Name = EECS and DEPT.Name = EECS). Declaring a semantic variable with an explicit domain requires the user, unlike in the global schema approach, to know the details of the local databases such as the entities and the attribute names desired.

If the user does not know such details, he can create a semantic variable with an implicit domain. For example, given the semantic declaration of DPT whose domain is declared to be DEP*, the system will take * as a wild card, try to determine how DPT should get translated for the databases involved by finding the entity or attribute names that satisfy the specification, and then translate the original query to local queries with appropriate identifiers for the different databases. In both explict and implicit domain cases, the semantic variable construct would work only if the specification, explicit or implicit, matches only those identifiers that the user wants. For example, in the explicit domain case, if another DB named the department entity as DPT, then this entity will be missed because DPT will not satisfy the specification DEP*. Furthermore, if there was an entity called DEPORT with an attribute called Name, then a query expression (DPT.Name = EECS) will also get translated into (DEPORT.Name = EECS), although that is not what the user had intended.

A semantic variable would be useful for KB integration provided that all the KBs use the same query language and that the caveats discussed above can somehow be avoided. It would allow the user to access objects and attributes in multiple KBs by simple syntactic stipulations. So as far as the name differences are concerned, the use of semantic variable can be convenient. But due to its syntactic nature (despite its misnomer -- "semantic" variable), its usefulness is pretty much confined to the name difference cases as we will see below

Granularity Differences

MRDSM offers no special construct for resolving granularity differences among databases. The user can, however, still deal with these differences by using the semantic variable construct that was described above. For example, suppose that STUDENT in DB1 is judged to be a supertype of GRAD-STUDENT in DB2 and that STUDENT's attribute ID is the same as GRAD-STUDENT's attribute StudentID. The user can again declare semantic variables S and GenericID with implicit domains *STUDENT and *ID respectively. The system will then try to determine how S and GenericID should get translated for the databases involved by finding the entity or attribute names that satisfy the specification, and then translate the original query to local queries with appropriate identifiers for the different databases. Although the user might get the same effect here as establishing a supertype relationship between STUDENT and GRAD-STUDENT (i.e. getting instances satisfying a given condition from both STUDENT and all of its subtypes even when they are distributed over different DBs), this solution is clearly not satisfactory. In this solution, the

knowledge of the supertype relation between the entities involved is not represented anywhere. It means that every time the user makes a query, he is responsible for figuring out all the entities in different DBs that he want to treat as subtypes of a given entity, the subtypes of those subtypes, etc. and explicitly list them in his query.

Value Restriction Differences

Suppose that the user wants to establish an equivalence between the Name attribute of DEPARTMENT in one DB and the ID attribute of DEPARTMENT in another DB. The name difference in the attributes, i.e. between Name and ID, can be resolved via semantic variables. But these two DBs might also associate different domain types with the attributes: eg. Alphanumeric and Integer. For example, the same department can be represented as EECS in one DB but as 6 in the other. MRDSM provides the construct Dynamic Attribute for setting up equivalence in such cases. The user can declare: *-attr-d* ID: *integer -define by D(*Name*)* = Dpt-ID-Dictionary, where Dpt-ID is a dynamic attribute that transforms the attribute Name and Dpt-ID-Dictionary is a dictionary containing the item-by-item translation from a value for Name to a value for Dpt-ID. For example, an entry in the table will contain the association between EECS and 6. The multiDB language processor will then translate any occurence of 6 in the second DB as EECS so that its value can be compared to or joined with the values from the first DB.

D in the above expression -define by *D(Name) stands for Dictionary*, but the user could also have other means of setting up the equivalence than through Dictionary. For example, if the type difference is between two scales, say Pound and Kilogram, then the user can declare *-define F(a)* = <arithmetical formula>, where the arithmetical formula converts one unit to the other. The user can also write an arbitrary procedure for converting a value to another and declare *-define P(a)* = <procedure name>. Since procedures can be as general as one makes, any attribute value type can be converted to another provided that the user can foresee all the possible values that need to be so converted. A problem with this construct is that often the user cannot determine in advance what all possible values are. However, this is not a problem unique to the present approach, but one also present in the other two approaches discussed.

This problem of integrating attribute values, as opposed to attributes or entities, also exists in KB integration, but is somewhat alleviated by the fact that in a KB the attribute values are themselves objects. In a DB, the only knowledge we have about the attribute values are often their data types such as INTEGER or STRING, which does not give us much knowledge about what they mean. In KB, since the attribute values are themselves objects that are related to other objects, we have some knowledge of what they mean. We can then use this knowledge for integration. For example, we may have the knowledge that at MIT the departments have numbers associated with them and that in particular, EECS department has the number 6. We may be able to use this knowledge to integrate the different types of attribute values. As we have seen when discussing Multibase, DBs can also have this knowledge represented in an auxilary DB. A difference is that an object in a KB, as opposed to a

data value in a DB, is related to many other objects, the knowledge of which may give a clue to how it can be compared to another object. But as long as the knowledge about an object is incomplete, this difficulty of attribute value integration will remain in KB as well.

Structural Differences

MDSL provides no construct for resolving structural differences. For example, there is no way to convert an attribute Department in one DB into an entity. However, the need to do so disappears in this approach. As we discussed in Section 4.1, the major reason for resolving structural differences among objects is to be able to construct an integrated schema (cf. Fig. 6). Since no global schema is constructed in this approach, there is no need for resolving structural differences. Of course, what the user has to do then is to explicitly enumerate the relevant attributes. For example, the user can no longer say simply "Give me all the information about EECS", but has to say "Give me all the values in any attribute with the name Dept* or Dpt or any attribute for an entity with name DEPT* or DPT". Also, since there is no semantic knowledge of how these attributes or entities which return values are related, it is not clear what the returned values would mean.

There are other constructs in MDSL such as Options and Implicit Join that are intended to help the user to access attributes when the user does not know exactly their names or where to find them. Options allow the user to have more control over when certain attributes should be accessed. For example, the user can say "Find me the *value of Department or Dpt or Dept attribute* but *only if* the same entity that the attribute belongs to has another attribute called Dep*Head." But, being based on pattern matching, they also suffer from the danger of not getting what the user intended or, even worse, from the danger of getting what he did not intend to. A major lesson that we can draw from this approach is that a purely syntactic approach presents many dangers and that a KB integration mechanism must provide some ways, such as virtual entity definition, to represent semantic relations among the entities in different KBs.

Evolution

Need for Consensus

There is no consensus required for this approach. Each user defines a semantic variable or a dynamic attribute as appropriate for his application and nobody else is affected by it. A shortcoming of this approach is, of course, that each user has to know at least some details of the local databases and that he has to make needed definitions every time he wants to make a query.

Need for Updates

Since there is no global schema, there is no need for updates resulting from the relation between a global schema and the local schemas. Instead, if a change is made to a local schema, the user has to

know the change if it affects him. It is arguable whether this delegation of the responsibility to the user is good or bad. A user, i.e. whoever needs to access database schema, has to be aware of any change in local schema even in a single database anyway. Of course, the difference is that now the user has to be aware of any change in any of the multiple databases he is accessing. But at least, he can quickly filter out what changes are relevant or not because his purpose is specific--say, developing a payroll system. The alternative of hiding these changes by indirection through a global schema shifts the burden of updates from the user to the system; but since the system has to foresee and try to accommodate all possible purposes or applications, it cannot be so selective. Any change in the local schema has to be reflected in the global schema.

Cost

There is no setup cost associated with constructing a global schema. However, one might say that there is a setup cost associated with every query the user makes insofar as he has to make appropriate definitions like declaring semantic variables. To do so, the user needs to know the multidatabase query language and must be familiar with the local database schemas-- at least the parts he wants to access. In other words, much of the burden in multiple database access is shifted from the system to the user. An advantage of doing so is that the user can set up definitions specifically for his purpose, and that the user need to know only those details needed for doing so. So the overall cost, i.e. the cost for the user and for the system taken together, is much less in this approach than in the other two approaches. An advantage of the global schema approach is, of course, that the user cost is smaller, although the overall cost, in particular the system cost, is much larger than this approach. The maintenance cost is smaller in this approach than in the global schema approach because it consists of simple syntactic translations. Again, the saving in the maintenance cost comes from having the user worry about the necessary details.

5. Lessons

What have we learned from the study of DB integration research as analyzed in this paper? I examined some of the constructs that DB integration researchers have proposed. I have indicated for each of the constructs discussed its adequacy with respect to both DB and KB integration. An important lesson is that we need to represent various relations among objects in a semantic not in a syntactic wy. The language used in Multibase to define virtual entities seems right. And the type and map derivation operators proposed by Heimbigner & McLeod seems the right kind of constructs to be used in such a language. Whether they are exactly right, necessary, or sufficient is a question that has yet to be explored. I have also identified some additional features present in KB but not in DB (Cf. Fig. 3). Some of the issues that arise due to these features-- eg. more sophisticated notion of equivalence required by non-atomic value restriction as well as non-atomic attribute values. However, I made no attempt to study full implications of these additional features for KB integration.

Undoutedly, these features will raise new sets of issues that KB integration research will have to deal with.

In studying these issues for KB integration, I believe that it is important to keep in mind some general lesssons that we can draw from the above analysis of DB integration research. I summarize these lessons below.

• *The global schema approach is not likely to work.* I have pointed out several problems with the global schema approach. It is very difficult to establish mappings among different entities or virtual entity definitions once and for all, independent of specific contexts or application purpose. There is a huge cost in constructing a global schema, both in consensus gathering and semantic difference resolutions required. Even when a global schema is established, it is likely to be so large as to be unmanageable. And it is difficult to introduce any change once the global schema is set up.

• *The locally global schema approach also has the problems associated with the global schema approach, though to a lesser extent.* Since the local DB needs to build a global schema only to the extent that it satifies its local needs, some of the problems with the global schema approach become less serious. For example, a local DB can set up mappings among objects specifically for its own application purpose. But other problems can still be serious. When an imported object changes its meaning or structure, other objects in the locally global schema might have to change. But there is no way to identify what needs to be changed.

• *A purely syntactic approach such as Multidatabase language is not likely to work either.* A problem with this approach is that we do not know what we are getting because it is based on pattern matching of identifiers. In particular, it does not allow us to represent the semantic relationships among the entities. It seems that we do need some facility for defining virtual entities to represent such relationship.

• *We need an incremental approach.* On the one hand, the global schema approach is not incremental because the global schema is set up once and for all and difficult to change thereafter. On the other hand, a shortcoming of the multidatabase query language approach results from the fact that it is not incremental in a different sense; Different users have to establish different definitions and mappings every time they make queries. There is no accumulation of what has been established. What we need is a mechanism that saves definitions in such a way that the user can reuse them or build new definitions on top of them.

• *We need a modular way of grouping the mappings and the definitions.* In part, the need for modularity stems from the need for the incremental approach discussed above. If the user wants to reuse or build on top of existing maps and definitions, then there should be some way of grouping those that are are used together. The need for modularity also stems from the need to support evolution. One reason why the global schema is likely to fail is that there is no modular way to incorporate

changes. One does not know which part of the global schema is affected by the present change We want a mechanism that allows us to group the maps and the definitions, and maintain dependency relations among those groups so as to accommodate both evolution and incrementality.

• *We need some way of establishing context-sensitive maps.* That the maps among different entities are context-sensitive or application-dependent is one reason why the global schema approach fails and why the multidatabase query language approach is appealing. In the latter, we need to worry about only those relations among the entities that matter for a given application or in fact for a given query But we also found that we need to represent and save such relations in a more meaningful way. Once such relations are available, we need a mechanism for choosing which of those relations should be used when.

• *We need an interactive approach to KB integration.* Most DB integration research assumes that the judgements of when entities in different DBs are equivalent, or more generally how they should be mapped, are externally supplied. As we have seen, there is a good reason for the assumption. Such judgements require much common sense (eg. in knowing that GRAD-STUDENT is a kind of STUDENT) as well as domain specific knowledge (eg. in knowing that AREA-EXAM is a kind of QUALIFYING-EXAM). Yet, establishing how different objects should be related is the hardest part in the integration task. So we want our KB integration mechanism to help us as much as it can in this process. One way is to try to construct a knowledge base that contains the required knowledge and use the knowledge to automatically establish the *equivalences*. But constructing such a knowledge base will be a laborious task, to say the least. Even when we have such a knowledge base, I am not sure whether we can trust the judgement based on it because, as noted before, how objects should be mapped often depends on the specific goal we have in mind. Another way to help the process of establishing the map is to consult such a KB that already exists-- i.e. the user. An important role of a KB integration mechanism can be suggesting possible mappings among the entities and then managing the consistency and the consequences of the mappings confirmed by the user.

These are the lessons I believe we can learn from DB integration research for KB integration. In the Appendix III, I present a scenario that illustrates a picture of a KB integration mechanism that has the desirable features discussed above.

Appendix I:
Examples of Knowledge Represented in KANDOR

KB1:

```
(knd-de frame area-exam exam primitive
        (all ID 1)                          ;ID must be unique
        (exists CommitteeMembers 3)  ;at least three members
        (exists Date 1)
        (exists Student 1))

;Identifier is a slot whose parent is Slot and whose value has to be
;Alphanumeric
(knd-de slot Identifier Slot (generic Alphanumeric))

(knd-de slot ID Identifier)         ;ID is a slot whose parent slot is Identifier
(knd-de slot CommitteMember Slot (generic FacultyMember))
(knd-de slot Date Slot (generic Date))
(knd-de slot Student Slot (generic Student))

(knd-de frame Student Person primitive
        (all ID 1)
        (exists Name 1)
        (exists Address 1))

(knd-de slot ID Identifier)
(knd-de slot Name Identifier)
(knd-de slot Address Identifier)

(knd-de individual Jin Student
        (ID 154525033)
        (Name Jin)
        (Addres "60 Wadsworth, Cambridge"))
...
```

KB2:

```
(knd-de frame GraduateStudent Person primitive
        (all ID# 1)
```

```
        (exists Name 1)
        (exists CoursesTaken 4)
        (exists Dept 1))

(knd-de slot StudentID Attribute (generic Alphanumeric))
(knd-de slot Name Attribute (generic String))
(knd-de slot CoursesTaken Attribute (generic Courses))
(knd-de slot Dept Attribute (generic Department))

(knd-de frame Department Group primitive
        (all Name 1)
        (exists DeptHead 1)
        (exists Office 1)
        (exists Phone 1))

(knd-de slot DeptHead DeptAttribute (generic FacultyMember))

(knd-de individual Jin GraduateStudent
        (StudentID 154-52-5033)
        (Name "Jintae Lee")
        (CoursesTaken (6841, 6999, ...))
        (Dept EECS))

...
```

A.2 Knowledge Base Query Functions

The next major grouping of KANDOR functions consists of functions to query the knowledge base in various ways. First, there are several functions that answer questions about individuals.

(knd-instance? '<individual> '<frame>)
is a predicate that determines whether <individual> is an instance of <frame>.

(knd-types '<individual>)
returns the most specific frames that <individual> is an instance of.

(knd-slot-values '<individual> '<slot>)
returns the slot fillers for <slot> asociated with <individual>.

(knd-slots-and-values '<individual>)
returns the slot and values pairs for <individual>.

Then, there are functions for slots.

(knd-slot-super? '<slot1> '<slot2>)
is a predicate that determines whether <slot1> is above <slot2> in the slot taxonomy.

`(kmd-subs '<slot>)`
returns the immediate offspring of <slot> in the slot taxonomy.

`(kmd-super '<slot>)`
returns the immediate parent of <slot>.

`(kmd-slot-tops)`
returns those slots with no parent in the slot taxonomy.

`(kmd-tr '<slot>)`
returns the type restriction of <slot> (a frame, an individual, or NIL).

The next several functions are concerned with frames.

`(kmd-instance? '<individual> '<frame>)`
is a predicate that determines whether <individual> is an instance of <frame>.

`(kmd-instances '<frame>)`
returns all instances of <frame> as a list.

`(kmd-filtered-instances '<frame> <form>)`
returns those instances of <frame> for which <form> evaluates, with **instance** bound to the instance, to non-NIL.

`(kmd-frame-super? '<frame1> '<frame2>)`
is a predicate that determines whether <frame1> subsumes <frame2>.

`(kmd-subs '<frame>)`
returns the immediate offspring of <frame> in the frame taxonomy.

`(kmd-supers '<frame>)`
returns the immediate parents of <frame>.

`(kmd-frame-tops)`
returns those frames with no parents in the frame taxonomy.

`(kmd-value-restrictions '<frame> '<slot>)`
returns the value restrictions for <slot> at <frame> (i.e., those frames that all slot fillers for the slot <slot> must be an instance of in instances of <frame>).

(kmd-min-restriction '<frame> '<slot> '<frame-or-value>)
returns the minimum number of slot fillers for <slot> at <frame> that must belong to or be <frame-or-value>.

(kmd max-restriction '<frame> '<slot>)
returns the maximum number of slot fillers for <slot> at <frame> (or nil if no max).

(kmd-restrictions '<frame>)
returns all the restrictions of <frame>.

(kmd-primitive? '<frame>)
determines whether <frame> is a primitive frame.

(kmd-decompositions '<frame>)
returns the decompositions that decompose <frame>.

(kmd-partial-classify '(<frame>*) '(<restriction>*))
performs partial classification by returning a list of the most specific frames in the frame taxonomy that subsume the conjunctions of the given frames and restrictions.

Finally, there are two functions for decompositions.

(kmd-decomposed '<decomp>)
returns the frame being decomposed.

(kmd-elements '<decomp>)
returns the elements of the decomposition.

REFERENCES

[Batini et.al. 86] Batini, C, Lenzerini, M., and Navathe, S.B. "A Comparative Analysis of Methodologies for Database Schema Integration." ACM Computing Survey, 18(4) Dec 1986, pp.323-364

[Brachman et.al. 83] Brachman, R., Fikes. R.E., & Levesque, H.J. "KRYPTON: A Functional Approach to Knowledge Representation." IEEE Computer, 16(10). Oct 1983, pp.67-73.

[Brachman & Levesque 86] Brachman, R. & H. Levesque. "What Makes a Knowledge Base Knowledgeable? A View of Databases from the Knowledge Level." In [Kerschberg 86]. pp.69-78.

[Brodie & Mylopoulos 86] Brodie, M.L. & Mylopoulos, J. (eds) On Knowledge Base Management Systems 1986 N.Y.:Springer-Verlag.

[Dayal & Hwang 84] Dayal, U. & Hwang, H. "View Definition and Generalization for Database Integration in a Multidatabase System." IEEE Trans. on Software Engineering. V. SE-10, No. 6. Nov 1984, pp.628-645.

[Heimbigner & McLeod 85] Heimbigner, D. & McLeod, D. "A Federated Architecture for Information Management." ACM Trans. on Office Information System. 3(3). July 1985, pp.253-278.

[Kerschberg 86] Kerschberg, L. (ed.) Expert Database Systems 1984 N.Y.:Springer-Verlag.

[Lyngbaek & McLeod 84] "Object Management in Distributed Information Systems." ACM Trans. on Office Information Systems 2(2), Apr 1984. pp.96-122.

[Litwin 84] "MALPHA: A Relational Multidatabase Manipulation Languages" IEEE COMPDEC, 1984. PP.86-93.

[Litwin & Abdellatif 86] Litwin, W. & Abdellatif, A. "Multidatabase Interoperability." IEEE Computer. Dec 1986, pp.10-18.

[Newell 81] Newell, A. "The Knowledge Level." The AI Magazine, 2(2). 1981 pp.1-20.

[Patel-Schneider 84] Patel-Schneider, P.F. "Small can be Beautiful in Knowledge Representation." Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems. 1984

ACKNOWLEDGEMENT

Many people gave me useful comments. Among them are Rick Lathrop, Jonathan Amsterdam, Karl Ulrich, David Kirsh, Franklyn Turbak, Kevin Crowston, Mark Ackerman, Wendy MacKay, and David Rosenblitt. Also, helpful were the comments from Professors Marvin Minsky, Ramesh Patil, and Thomas Malone. I thank all of them.

DISTRIBUTION:

Defense Technical Information Center

Computer Sciences Division
ONR, Code 1133

Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Code 5510

Dr. A.L. Slafkosky
Headquarters, U.S. Marine Corps (RD-1)

Psychological Sciences Division
ONR, Code 1142PT

Applied Research & Technology
ONR, Code 12

Dept. of the Navy
Naval Sea Systems Command
NAVSEA 90

Dr. Charles Schoman
David Taylor Naval Ship R&D Center
NSRDC 18

END
FILMED
7-89
DTIC