5/16/88 *

AN OBJECT-ORIENTED APPROACH TOWARDS ENHANCING LOGICAL CONNECTIVITY IN A DISTRIBUTED DATABASE ENVIRONMENT

by
David Collins Horton III

B.A., Economics and Physics
Amherst College
(1982)

Submitted to the Sloan School of Management
in Partial Fulfillment of
the Requirements of the Degree of
Master of Science in Management

at the

Massachusetts Institute of Technology

May 1988

Signature of Author ....
Alfred P. Sloan School of Management
May 13, 1988

Certified by ....
Y. Richard Wang
Assistant Professor, Management Science
Thesis Supervisor

Accepted by ....
Jeffrey A. Barks
Associate Dean, Master's and Bachelor's Programs

AN OBJECT-ORIENTED APPROACH TOWARDS ENHANCING LOGICAL CONNECTIVITY IN A DISTRIBUTED DATABASE ENVIRONMENT

by
David Collins Horton III

B.A., Economics and Physics
Amherst College
(1982)

Submitted to the Sloan School of Management
in Partial Fulfillment of
the Requirements of the Degree of
Master of Science in Management

ABSTRACT

The advent of distributed database systems brings with it the challenge of making logical, as well as physical, connections between disparate systems. Taking advantage of the "information explosion" and lower hardware costs means reconciling the semantics of different system environments and administrators. Traditional database systems have no tools for capturing these kinds of knowledge, while traditional AI systems don't feature flexible access to "real-world" databases. This thesis presents the design of a Composite Information System Tool Kit (CIS/TK) which is being developed to bridge this gap. It is found that an object-oriented approach can successfully embed semantic and heuristic knowledge in a distributed database environment to deal with problems such as inferencing, mapping data between different domains and assembling complete information about entities from extremely disjoint distributed databases.

Thesis Supervisor: Dr. Y. Richard Wang
Title: Assistant Professor of Management Science

ACKNOWLEDGEMENTS

I'd like to express my gratitude to everyone who assisted me in this work, and to all those who were working on related topics. First and foremost, thanks to my advisor, Rich Wang, who always made himself available and was willing to meet at any time to discuss new ideas. His help, and his determination that this project should succeed, were the driving forces in my work, as well as many others. In the same vein, I hasten to thank T.K. Wong for his effort, his time and his code (of course); but most of all for his thought in our design sessions. Time and again he was able to focus on the pertinent design aspects and prevent me from pursuing directions that would have proved fruitless. I may never understand why an undergraduate would work so hard, but I certainly won't begrudge my good luck. I also owe a large debt of gratitude to Tammy Son, without whom I couldn't have gotten *anything* done but who, more importantly, was always fun to be around, even when we came up with new and exciting ways to make her life sheer hell. Believe me Tammy, that landscape graphic wasn't my idea !

Thanks also to Prof. Stuart Madnick, my reader, for his time and thought. His criticisms and ideas frequently provided the force to develop the final, and most critical, design elements. Many thanks to Larry Kooper, and his PAS team, for their work which kept me cognizant of the needs of application users like the Placement Office, and whose intensity forced me to try to keep up.

Finally, thanks to all who put up with me and tried to keep me sane during this period, especially my parents, Sarah, and William Keefe Ltd.--may he one day break out of his self-imposed existential cage.

And, of course, thanks most of all to the people at the Royal East.

CONTENTS

Introduction

This thesis explores how an object-oriented platform can be used to develop a set of tools designed to enhance logical connectivity in a distributed database environment. A prototype, called the Composite Information System Tool Kit (CIS/TK) is currently being developed to experiment and codify successful approachs to the problem of logical connectivity. The thesis is organized into four reports followed by a section of short concluding remarks. Each report focuses on a separate aspect of CIS/TK. The first report presents an overview of the applications of CIS systems and discusses the goals of the CIS/TK system, in particular. The next report examines the specific nature of query processing in CIS/TK, detailing the implementation of a layered approach towards query processing. The third report examines the basic logical connectivity problem of translating data values between different domains, as is frequently needed in order to join data in a distributed environment. The fourth report details a technique called attribute subsetting which is used to join information between databases where no primary-foreign key join is available. Finally, a concluding chapter examines the direction and focus of future work.

A Tool Kit for Composite Information Systems (CIS/TK): Research Overview

I. Introduction

The rapidly increasing complexity, interdependence, and competition in the global market over the last two decades has profoundly impacted how corporations operate and how they (re)align their information technology and competitive strategies in the marketplace. This alignment has accelerated demands for more effective information management for both decision-making, operational efficiency, and new product and services. To meet these demands, the information industry has made significant advances in the price, speed performance, capacity, and capabilities of database technology. Today, commercial Data Base Management System (DBMS) products are widely installed by corporations. Homogeneous distributed DBMSs such as Relational Technology's INGRES* and ORACLE's SQL*STAR are now becoming commercially available. Furthermore, experimental heterogeneous DBMSs such as CCA's MULTIBASE have also been introduced.

In exploiting the DBMS technology, it has become increasingly evident that many important applications require multiple independent disparate databases to work together within and/or across organizational boundaries in order to increase productivity. We refer to this type of systems as *Composite Information Systems* (CIS). Although many Composite Information Systems exist today, they are in reality a combination of human operators and computer systems. The human intervention required to interface multiple independent databases implies that it is an expensive, time-consuming, and error-prone process. It would be advantageous if the human operator component could be automated. In attempting to automate the human operator component, we have identified three important techniques: table lookup, functional mapping, and heuristic reasoning. They are exemplified below using a multiple tour guide case study [Madnick and Wang, *Facilitating Connectivity in Composite Information Systems*, to appear in Database]:

(1) *Table lookup*: Many syntax problems can be resolved through table lookup. For example, the entity "amenity" in the Massachusetts Spirit tour guide database is called "facility" in the FODOR tour guide; similarly, the instance "A/C" in FODOR is called "air conditioning" in the Massachusetts Spirit.

(2) *Functional mapping*: Procedures are useful to encode formulae and facts. For example, in FODOR *expensive* means, among other criteria, "bath or shower in each room, restaurants, TV, phone, attractive furnishings, heating, and A/C." Since the meaning of *expensive* is not stored as part of the relations, a procedure is needed to encode the information.

(3) *Heuristic reasoning*: There are many rules of thumb which do not fit either tables or procedures very well. For example, if the lodging type is a *motel*, then it would be reasonable to encode a heuristic rule stating that free parking is available. Alternatively, if a lodging's location is in the Boston Back Bay area (from zip code 02116), and the lodging is rated as $$$, then valet parking is very likely to be available.

Faced with these problems, we have found it effective to incorporate Artificial Intelligence (AI) technology as part of the solution. AI technology developed over the past 30 years has proven to be very useful in capturing rules of thumb and making rule-based inferences. By integrating the information sharing capability of DBMS technology and the knowledge processing power of AI technology, multiple independent disparate databases may be accessed in concert with minimum human intervention.

In addressing the integration of AI and DBMS technologies, Albert, Chern, and Sears [Forward of Topics in Information Systems On Knowledge Base Management Systems, Brodie and Mylopoulos, ed., 1986] have suggested that:

> "The time has come to face the complex research problems that must be solved before we can design and implement real, large scale software systems that depend on knowledge-based processing. In the long term, research is needed to find ways for knowledge-based system technology to support database systems and vice versa. In the near term, research is needed to develop tools that support the design and development of systems that require an integrated set of knowledge base and database system tools."

The *Tool Kit for Composite Information Systems* (CIS/TK) is an innovative research prototype being developed at the MIT Sloan School of Management for providing such an integrated set of knowledge base and database system tools. It is being implemented in the UNIX environment to take advantage of its portability across disparate hardware and multi-programming capability for accessing multiple disparate remote databases in concert.

II. The Tool Kit for Composite Information Systems (CIS/TK)

The CIS/TK ensemble can be viewed as a *Knowledge and Information Delivery System* (KIDS), as depicted in Figure 1, which has four functional components: knowledge processing, information sharing, physical and logical connectivity, and user interfaces. Specifically, it consists of the following four subsystems:

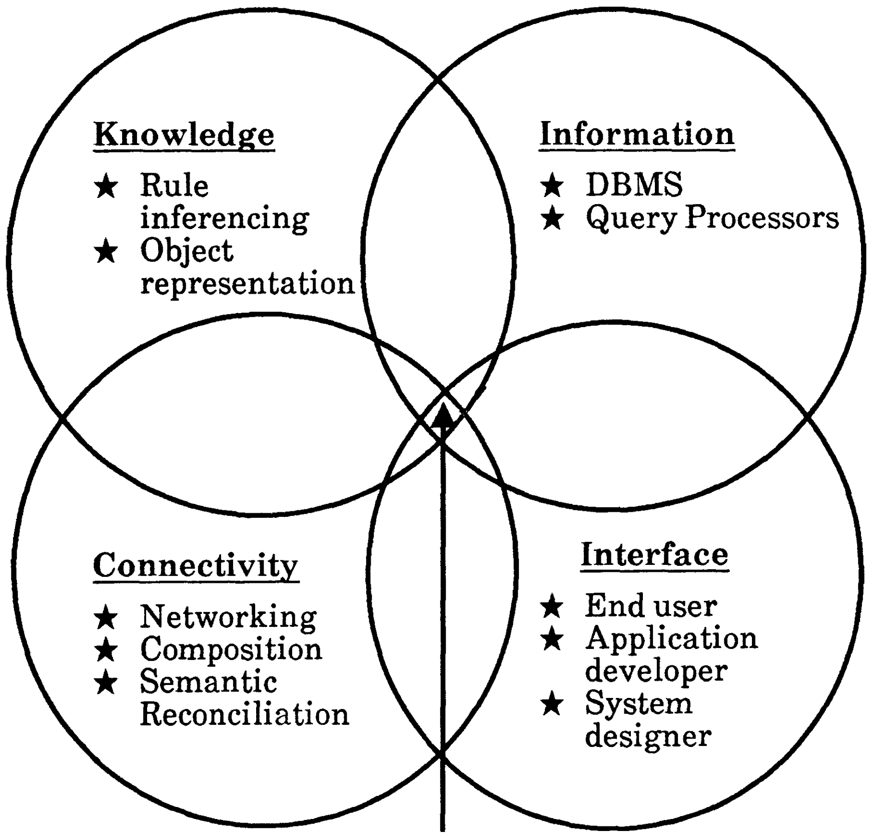


Figure 1 Knowledge and Information Delivery Systems [KIDS]

(1) Knowledge Processing

An enhanced version of the Knowledge-Object REpresentation Language [Levine, 1987] which facilitates an object-oriented approach and rule-based inferencing mechanism. Implemented in Common Lisp, it forms the underpinnings of CIS/TK. The significance of the subsystem is threefold: (1) it gives us the capability to evolve the code for experimenting and developing innovative concepts; (2) it provides the required knowledge representation and reasoning capabilities for knowledge-based processing in the heterogeneous distributed DBMS environment; and (3) it is very simple to interface the code with off-the-shelf software products (e.g., EXPRESS, ORACLE, and INFORMIX) through the I/O redirection and piping capability inherent in the UNIX environment. The reader is referred to Levine [Master's Thesis, Electrical Engineering and Computer Science, MIT, May 1987] and Wong [*Enhancing The CIS/TK Rule Inferencing Capabilities*, Sloan School of Management, MIT, May 1988 (expected)] for a detailed description of the knowledge processing component.

(2) Information sharing & Physical Connectivity

A multi-layered query processing architecture for processing end-user queries. This architecture will be examined in detail in chapter 2. The architecture consists of an application query processor, a global query processor, and a local query processor and a DBMS query processor for each DBMS in the CIS. The query processors are key to the integration of disparate databases in a heterogeneous environment where multiple-vendor hardware and DBMSs may need to be accessed in concert. This subsystem corresponds to the shared information component of KIDS with the exception that the local query processors are also responsible for physical connectivity.

(3) User Interfaces

A set of user interfaces for building global schemata, application models, and application model queries. The Global Schema Builder facilitates the database designers and administrators to create global schemata for integrating disparate databases. The Application Model Builder facilitates the application developer to create application models. The User Query Builder facilitates the end-user to build queries given an application model and its corresponding global schema. The reader is referred to Levine [*The Design and Implementation of the CIS/TK*

User Interface Builders, Sloan School of Management, MIT, May 1988 (expected)] for the details.

(4) Logical Connectivity

A set of special facilities for achieving logical connectivity; for example, *domain value translation*, for mapping values between different representation schemes, and *instance identification* through attribute subsetting. Chapters 3 and 4 detail how these facilities are implemented in CIS/TK.

Taken together, these four subsystems comprise a **KIDS** for delivering timely knowledge and information in a diversity of situations. CIS/TK can be applied to a diversity of situations, as discussed below.

III. Applications of CIS/TK

Four categories of situations where KIDS can be strategically advantageous are summarized below:

(1) Inter-organizational - which involve two or more separate organizations (e.g., direct connection between production planning system in one company and order entry system in another company).

(2) Inter-divisional - which involves two or more divisions within a firm (e.g., corporate-wide coordinated purchasing).

(3) Inter-product - which involves the development of sophisticated information services by combining simpler services (e.g., a cash management account that combines brokerage services, checks, credit card, and savings account features).

(4) Inter-model - which involves combining separate models to make more comprehensive models (e.g., combine economic forecasting model with optimal distribution model to analyze the impact of economic conditions on distribution).

As an example, consider the Composite Information System for the Sloan placement office, as illustrated in Figure 3. The CIS spans five systems in four organizations: (1) the student database and the interview database are located in the Sloan School; (2) the alumni database is available in the alumni office; (3) the *recent news* is accessed by dialing into the Reuters' textline database; and (4) the *recent financial information* is accessed through the I.P. Sharp's Disclosure II database. In order to find companies interviewing at Sloan that are *auto manufacturers,* alumni/students from these companies, and recent information about these companies, all the five databases in the four organizations need to be accessed. CIS/TK can be applied to facilitate this process through its query processor subsystem. Moreover, its knowledge processing component can be employed to perform complex heuristic reasoning. The reader is referred to Kooper [*A Composite Information System for the Sloan Placement Office*, Master's Thesis, Sloan School of Management, MIT, May 1988 (expected)] for the details.

IV. Summary

We have presented a broad overview of a Tool Kit for Composite Information Systems (CIS/TK). The CIS/TK ensemble is a unique and innovative, unique, cutting-edge system for delivering timely knowledge and information in an inter-organizational setting. In the rapidly changing, complex, and competitive global market, the capability to dynamically (re)align corporate strategy with information technology (IT) in the organizational context is a critical issue facing the IS

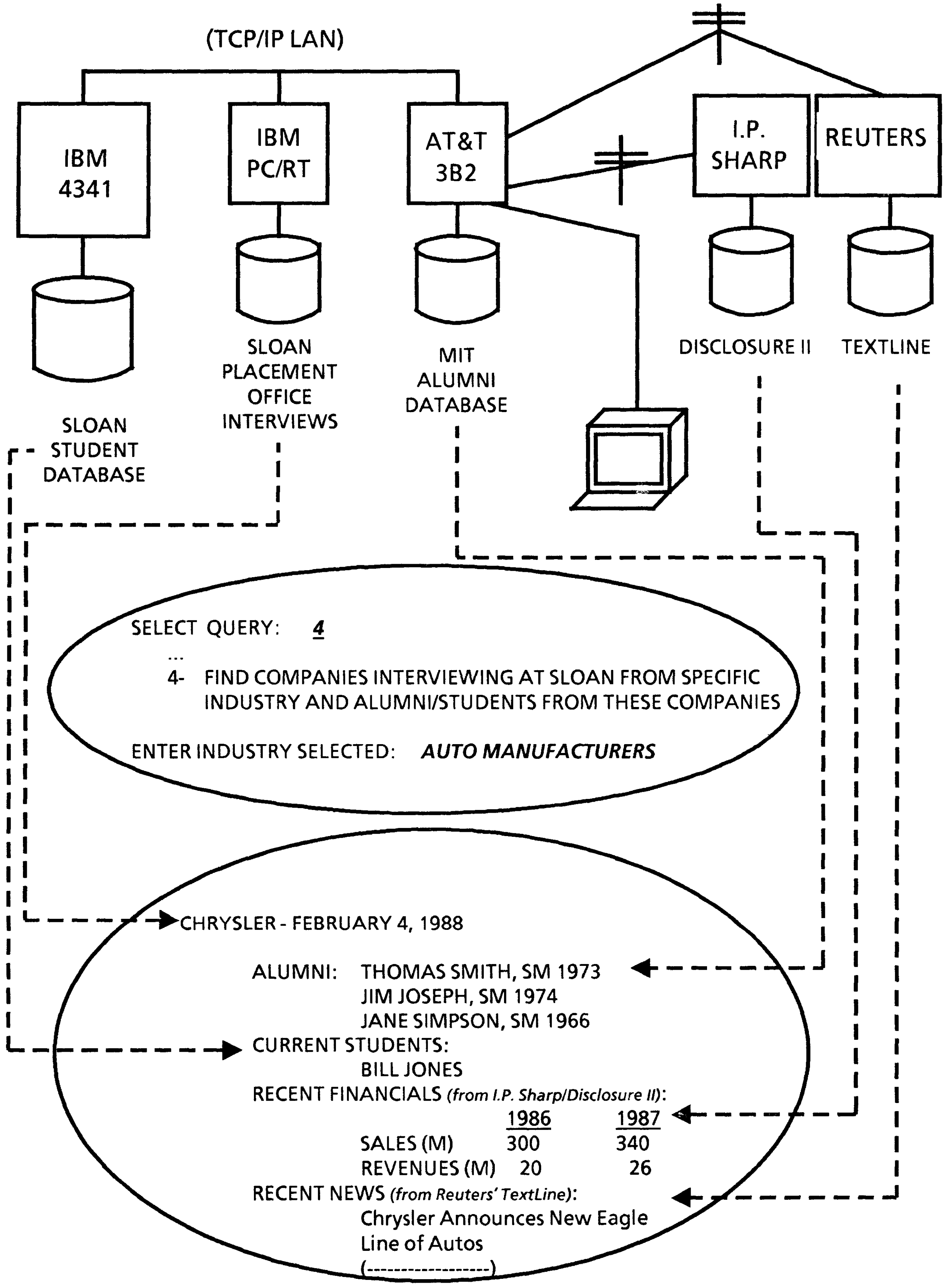

(TCP/IP LAN)
IBM 4341
IBM PC/RT
AT&T 3B2
I.P. SHARP
REUTERS
SLOAN STUDENT DATABASE
SLOAN PLACEMENT OFFICE INTERVIEWS
MIT ALUMNI DATABASE
DISCLOSURE II
TEXTLINE
SELECT QUERY: 4
...
4- FIND COMPANIES INTERVIEWING AT SLOAN FROM SPECIFIC INDUSTRY AND ALUMNI/STUDENTS FROM THESE COMPANIES
ENTER INDUSTRY SELECTED: AUTO MANUFACTURERS
CHRYSLER - FEBRUARY 4, 1988
ALUMNI: THOMAS SMITH, SM 1973
JIM JOSEPH, SM 1974
JANE SIMPSON, SM 1966
CURRENT STUDENTS:
BILL JONES
RECENT FINANCIALS (from I.P. Sharp/Disclosure II):
1986 1987
SALES (M) 300 340
REVENUES (M) 20 26
RECENT NEWS (from Reuters' TextLine):
Chrysler Announces New Eagle Line of Autos
(------------------)

Figure 3 A CIS for the Sloan Placement Office

executive. The CIS/TK research is aimed at providing such a dynamic IT platform for supporting knowledge and information intensive applications.

The Design of the CIS/TK Query Processor Architecture

The *Tool Kit for Composite Information Systems* (CIS/TK) is a research prototype being developed at the MIT Sloan School of Management for providing an integrated set of knowledge base and database system tools. Developed in the UNIX environment to take advantage of its portability across disparate hardware and multi-programming capability for accessing disparate remote databases in concert, the CIS/TK ensemble consists of the following subsystems:

(1) An enhanced version of the Knowledge-Object REpresentation Language (KOREL) [Levine, 1987] which facilitates a frame-based knowledge representation and rule-based inferencing mechanism

(2) A query processor architecture for processing end-user queries. The architecture consists of an application query processor, a global query processor, and a pair of local query processor and DBMS query processor for each DBMS in the CIS.

(3) A set of user interfaces for building global schemata, application models, and application model queries. The Global Schema Builder facilitates the Database administrator to create global schemata for integrating disparate databases. The Application Model Builder facilitates the application developer to create application models. The User Query Builder facilitates the end-user to build queries given an application model and its corresponding global schema.

(4) A set of special facilities for achieving logical connectivity; for example, *conflict resolution* of incompatible information through credibility analysis and concept inferencing, and *instance identification* through attribute subsetting and name recognition.

Three classes of users will interact with the CIS/TK: (1) *Database administrators* who create and implement global schemata based on the underlying, distributed local schemata; (2) *applications designers* who create and implement application models based on an underlying global schema; and (3) *end-users* who generate queries based on the application model.

End-users can generate queries in one of two fashions. Queries can be pre-built by the application designer and stored directly into the application objects. A user-friendly front-end then presents these queries to the end-user for easy selection. This procedure allows even the most unsophisticated end-users to be supported for commonly-occurring queries. Alternatively, the end-user can generate ad-hoc queries by directly formulating the query himself. In the future, an SQL-type parser will be implemented. Currently, the query must be formulated in the application model query syntax.

The design decisions and implementation details of each of the subsystems will be documented in a series of CIS/TK reports. We focus on the CIS/TK query processor architecture in this report.

I. The CIS/TK Query Processors

Before discussing the query processor architecture it is necessary to define the terminology used. End-user queries interact with a set of objects that make up the Application Model (AM). The AM is one application designer's view of an underlying Global Schema (GS). The AM will be described in traditional object-oriented programming terms -- that is, as a set of objects having both slots and methods. AM objects may have superiors and subtypes, but other types of relations will be represented in these objects as well. For example, students and transcripts are objects in a student AM. The GS, however, represents database concepts. Thus, a GS will be described as a set of entities, each having attributes and relationships (1 to n, n to 1, and m to n) with other entities. Although the GS entities will be implemented as objects, we will continue to refer to a GS in database terminology of entities, attributes and relationships.

A layered architecture for processing end-user queries is proposed, consisting of the following four query processors, as shown in Figure 1:

(1) Application Query Processor (AQP)

(2) Global Query Processor (GQP)

(3) Local Query Processors (LQPs)

(4) DBMS Query Processors (DQPs)

The architecture is presented as follows: first, the interfaces between each of the four query processors are examined. For each query processor, the syntax for inbound and outbound messages types is laid out. Moreover, the Global Schema Manager (GSM) and the Application Model Manager (AMM) which support the GQP and AQP respectively are also examined in the context of query processing. Next, two specific application query scenarios are developed to further illustrate how queries are decomposed and handled by each query processor. Finally, the innards of the AQP and GQP are examined in light of a specific query. The reader is referred to Champlin [1988] for the design decisions and implementation details of the LQP.

The local schemata, related global schema and application model that are shown in Figures 2, Figure 3, and Figure 4 will be used throughout to support the query examples. These are the local and global schema for a student CIS system.

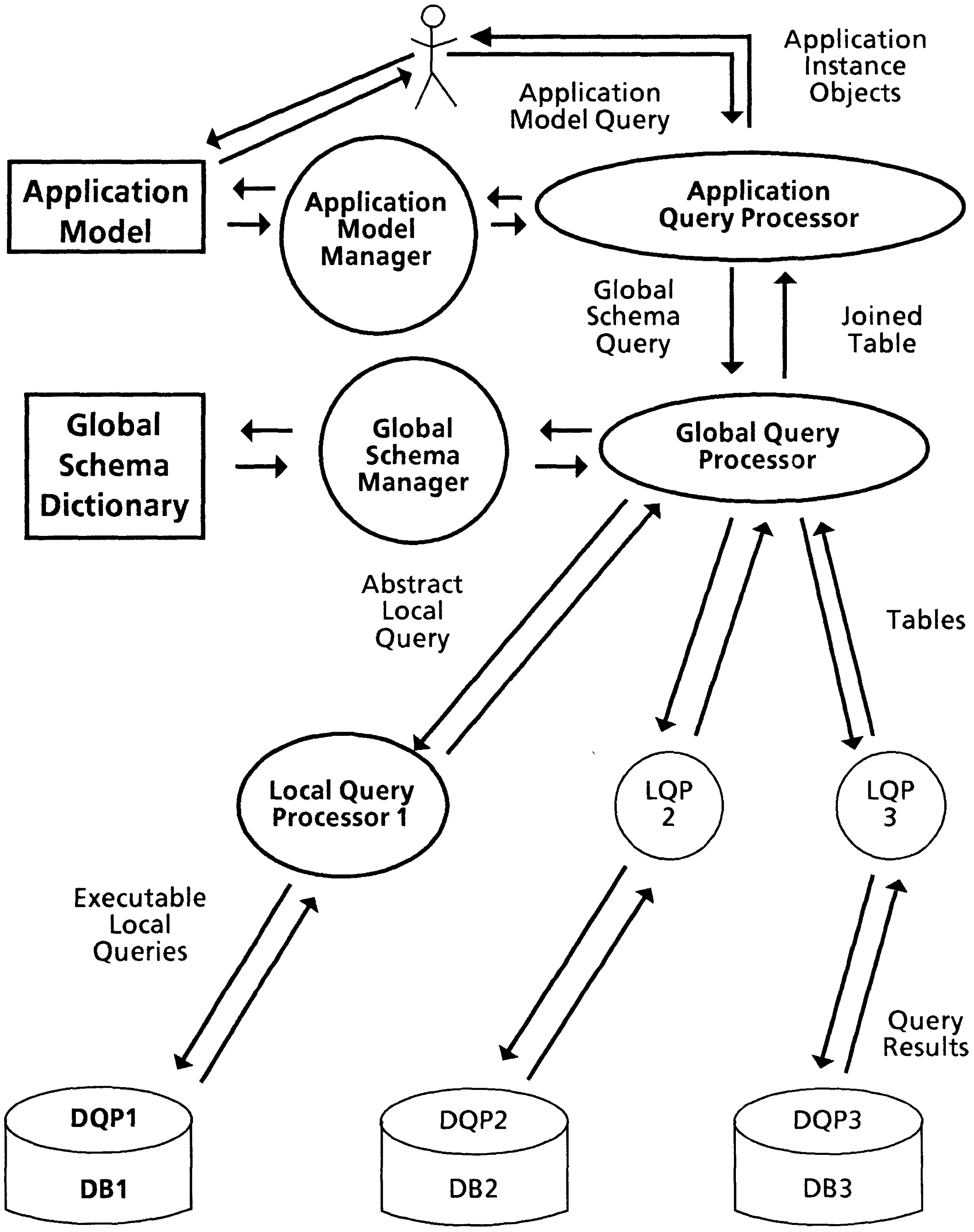


Figure 1 The CIS/TK Query Processor Architecture

***Student db* (STUDENTDB)**

Grade tb

Sname	ID	Average	Club

Major tb

Name	Student-ID	Concentration

***Course-hist db* (COURSEHDB)**

Course-hist tb

Name	Course #	Prof.	Grade

***Course-info db* (COURSEIDB)**

Course-info tb

Course-num	Credits	Desc.

***Club db* (CLUBDB)**

Club tb

Cname	# members	Funding

Figure 2 Local Schemata of Four Separate Student Databases

The corresponding object specifications for the global schema and application model are also shown in Figure 5 and Figure 6 respectively.

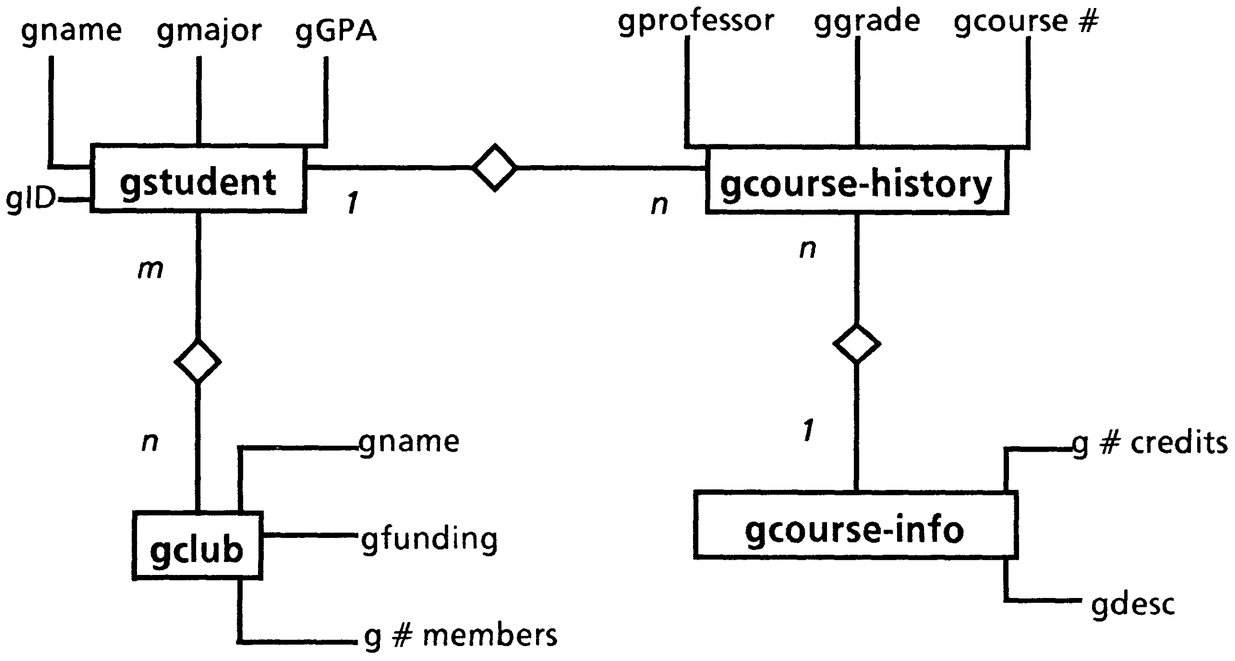


Figure 3 The Student Global Schema

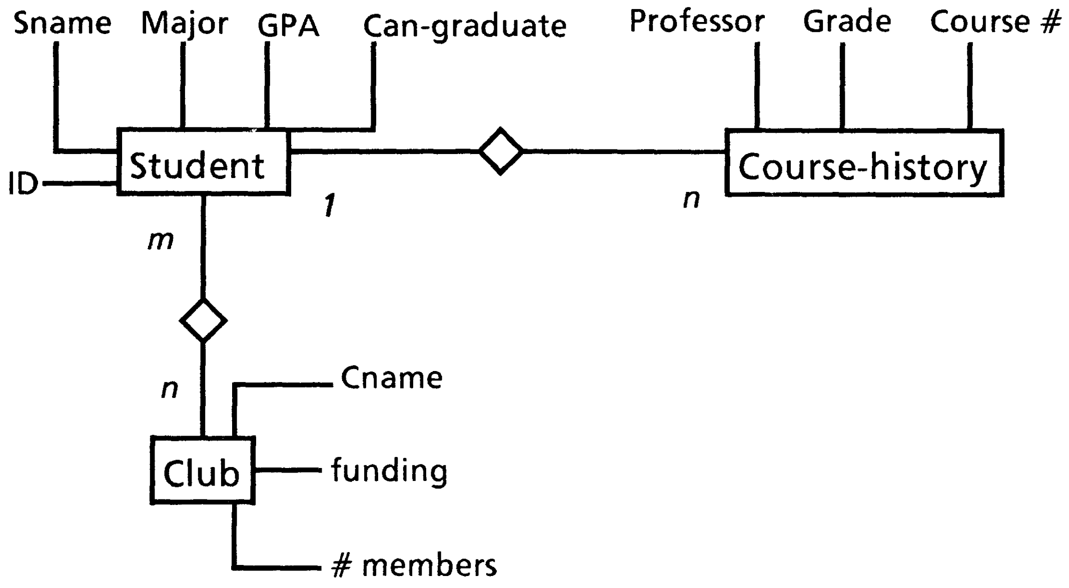


Figure 4 A Student Application Model

Global Schema

Object	Slot	Value
mit-gschema	DB__loc	(BOOT DB__loctb)
	attrib__map	(BOOT attrib__maptb)
	entity__list	(gstudent gclub gcourse-history gcourse-info)
	join__keys	((STUDENTDB gradetb (gname gid)) (STUDENTDB majortb (gname gid)))
gstudent	gname	((STUDENTDB gradetb sname) (STUDENTDB majortb name))
	gmajor	((STUDENTDB majortb concentration))
	ggpa	((STUDENTDB gradetb average))
	gid	((STUDENTDB gradetb id) (STUDENTDB majortb student-id))
	join-m-to-n	((gclub (gstudent gname)))
	join-1-to-n	((gcourse-history (gstudent gname)))
gclub	gname	((CLUBDB clubtb cname))
	gfunding	((CLUBDB clubtb funding))
	g#members	((CLUBDB clubtb #members))
	join-m-to-n	((gstudent (gstudent gname)))
gcourse-history	gprofessor	((COURSEHDB course-listtb prof))
	ggrade	((COURSEHDB course-listtb grade))
	gcourse#	((COURSEHDB course-listtb course#))
	join-n-to-1	((gstudent (gstudent gname)) (gcourse-info (gcourse-history gcourse#)))
gcourse-info	g#credits	(COURSEIDB course-infotb credits))
	gdesc	(COURSEIDB course-infotb desc))
	join-1-to-n	((gcourse-history (gcourse-history gcourse#)))

Figure 5 The Object Specification of the Student Global Schema

A few notes about these objects are in order. First, the DB__loc and attrib__map slots of the global schema object are created and used by the builder interface to the Global Schema (for details refer to Levine[1988]). They simply

Application Model

Object	Slot	Value
user-app-1	objects	(student club course-history)
	gschema	(mit-gschema)
student	app-model	(user-app-1)
	sname	(gstudent gname)
	major	(gstudent gmajor)
	gpa	(gstudent ggpa)
	id	(gstudent gid)
	can-graduate	(graduation-rule-set) ; in RULE facet
	related-to	((club (student sname)) (course-history (student sname)))
club	app-model	(user-appl)
	cname	(gclub gname)
	funding	(gclub gfunding)
	#members	(gclub g#members)
	related-to	((student (student sname)))
course-history	app-model	(user-appl)
	professor	(gcourse-history gprofessor)
	grade	(gcourse-history ggrade)
	course#	(gcourse-history gcourse#)
	related-to	((student (student gname)))

Figure 6 The Object Specification of A Student Application Model

point to two tables stored on the local machine: one containing information on all the databases in the system (i.e. DBMS type and machine location) and the other containing information on the entities that constitute the global schema -- in particular information concerning how each attribute is named in each of the local schemas. Tables provide the most logical structure for storing and updating this information. At load time, the attrib_map table is used to create objects for each of the entities in the global schema. A slot is created for each attribute of that entity and its value is a symbolic pointer to the LQPs which can satisfy data requests for that attribute, along with the table and column name. the join-1-to-n, n-to-1 and m-to-n slots are created as needed to model the joins reflected in the global schema. The value of these join slots is the entity to which it is joined, paired with the entity/attribute combination on which the join is made at the local schema level. For instance, the local schema (Figure 2) shows that joins between students and their course histories must be made on student name. The value of the join-1-to-n slot in the gstudent object reflects both the relationship between gstudent and gcourse-history, and the linking attribute. This information must be stored in order to be able to link student instances with their appropriate course histories at the application model level.

The objects in the application model are clearly similar to the objects in the global schema. The application model object itself contains only information about its related global schema and a list of objects which constitute the application model. Each of the objects contains a slot which may point to an entity/attribute combination in the global schema, or may contain a set of rules for inferencing. The related-to slot indicates the relationships between objects, as well as the slot value which joins them in the underlying local schema.

With the specifications in place, we now examine the interfaces between AQP, GQP, LQP, and DQP as well as the corresponding GSM and AMM.

The Application Query Processor (AQP)

The AQP is an object which has a query method that handles messages referring to objects in the AM. These messages originate from the end-user, either by selection from the list of pre-built queries stored in the AM or by direct formulation by the end-user himself. This message is referred to as an Application Model Query (AM query).

The task of the AQP is to instantiate applications objects as requested by the AM query. It will do this by translating the AM query into appropriate Global Schema queries (GS queries) and sending them on to the GQP as a message. Upon return, it will translate the returned results (which are in LISP list structure) from the global schema format to the application model format and create instance objects. These instance objects will then be available to the end-user for display and other types of manipulation, such as rule-based inferencing. Thus, the AQP handles all mapping issues between the AM and the GS. The knowledge needed to perform this mapping is encoded in the AM. It is provided to the AQP by the Application Model Manager.

Message format accepted by the AQP:

```
(send-message 'AQP :query
          ((object1 (slot1 ... slotn)
                 ((constraint1) .. (constraintn)))
           .
           .
           .
           (objectn (slot1 ... slotn)
                 ((constraint1) .. (constraintn))))
```

Where constraint is represented as:

```
    (condition  slot  value)
```

i.e.:

```
(send-message 'AQP :query (student (name major gpa)
                ((= major "physics")
                 (> gpa 3.0)))
```

is equivalent to an SQL query :

```
select name, major,gpa
    from global-schema
    where major = "physics" and gpa> 3.0
```

Application Model Manager (AMM)

The job of the AMM is to provide information about an Application Model to the AQP. The AMM accepts only one message type: *get-GS-nomenclature.* It is a request for global schema information about a single object/slot combination in the AM. For example, given an AM and GS as shown in Figures 2 and 3, if the AM manager were passed the object /slot combination (student name) it would return (gstudent gname) because that is where the information is referenced in the GS.

Message format accepted by the AM Manager:

```
(send-message 'AMM :get-GS-nomenclature (object slot))
```

Global Query Processor (GQP)

The GQP accepts a message from the AQP which refers to entities and attributes that are defined in the GS. It first decomposes the GS query into the appropriate Abstract Local queries (AL queries) which use the local schema nomenclature. One of its significant tasks, therefore, is to decide how the results will be joined -- by a primary-foreign key join, or by attribute subsetting. Based on this decision, the AL queries will be formed in such a way as to ensure that any columns necessary for joining are requested along with those columns needed to satisfy the original GS query. These AL queries are then sent as messages to the appropriate LQP. Upon return local attribute names are re-mapped to the global schema and all necessary joins are performed by the GQP. The resulting list structure is then returned to the AQP. In order to perform these activities the GQP will need to query the Global Schema Manager (GSM) about:

(1) Which DBs and tables contain data for a given attribute in the global schema.

(2) How the attributes are represented in those tables in which they appear (i.e. column names).

(3) Which attributes of an entity are candidate keys, and thus can be used to perform primary-foreign key joins if entities are split across multiple tables.

Message format accepted by the GQP:

```
(send-message 'GQP :query (entity (attrib1 attrib2..attribn)
                ((constraint1) .. (constraintn)))
```

Where constraints can be represented as:

```
    (condition (entity attribute) value)

i.e.( = (course-taken course#) 15.579)
```

If, however, the entity in the constraint is the same as the first argument to the :query method then the constraint can simply be stated as:

```
    (condition attribute value)

i.e.:

(send-message 'GQP (course-taken (course# course-name)
                  (( = course# 15.579)
                   (> grade B)
                   ( = (student name) "sam")))
```

Global Schema Manager

The GS manager provides the GQP with the information about the GS attributes that is needed to send the appropriate LQP queries. The GS manager accepts three types of messages from the GQP:

```
(1). get-LQP-location ((entity attribute) ... (entity attribute))
        returns:
    ((entity attribute ((LQP tablename) ... (LQP tablename))) ...
     (entity attribute ((LQP tablename) ... (LQP tablename))))
```

For each entity/attribute pair the get-LQP-location method will return the LQPs which can access data for that pair. The GQP uses this information to determine where to access data for a given attribute. For example,

```
    (send-message 'GSM :get-LQP-location '((gstudent gmajor)(gstudent gname)))
```

would return

```
((gstudent gmajor ((studentdb majortb)))(gstudent gname
                ((studentdb majortb)(studentdb gradetb))))
```

(2). get-LQP-nomenclature (entity attribute LQP tablename)

returns:

column-name

For each GS attribute in the named table the GS manager returns the actual column name in that table. For example,

(send-message 'GSM :get-LQP-nomenclature '(gstudent gmajor MAJOR majortb))

would return:

concentration

(3). get-LQP-join-keys ((LQP-1 tablename) (LQP-2 tablename))

returns:

((entity attribute) ... (entity attribute))

Given the two database/table pairs, the GSM returns a list of the attributes which could be used to join entities which are split between two tables. In relational database terminology, this is a list of the candidate keys which are common to both tables. The list is returned in the GS nomenclature. For example,

```
(send-message 'GSM :get-join-keys '((GRADEDB gradetb) (MAJORDB majortb)))
```

would return:

```
((gstudent gname) (gstudent gid))
```

since both student name and id could be used to uniquely identify entities in gradetb and majortb.

We now turn our attention to the Local Query Processor.

Local Query Processor (LQP)

The LQP objects each accept a query on a single database-machine-DBMS instance. It translates the query into an executable query for that DBMS, establishes a connection with that machine and sends the query. The physical retrieval of data is then performed by the DBMS query processor (i.e. ORACLE, INFORMIX etc.). Upon return the LQP strips extraneous information from the resulting output stream and translates the results into LISP list format which is then passed back to the GQP. As mentioned earlier, the reader is referred to Champlin [1988] for the design decisions and implementation details of the LQP.

Message format accepted by the LQP to get data from the actual DBMS:

```
(send-message LQP :get-data '(table (col1 ... col2)
              ((constraint1) .. (constraintn))))
```

i.e.:

```
(send-message ORACLE1 :get-data  '(gradetb
              (sname concentration)
              ((= concentration "physics"))))

(send-message ORACLE2 :get-data '(majortb
              (name average)
              ((> average 3.0))))
```

Note that the GQP needs to know the name of the LQP handler for a given database, not the name of the database itself. How this information is made available to the GQP will be covered later, however, in the examples to come assume that the LQPs for the for the databases in the local schemas are named as follows: STUDENTDB is the LQP for studentdb, CLUBDB is the LQP for clubdb, COURSEHDB is the LQP for course-historydb and COURSEIDB is the LQP for course-infodb.

DBMS Query Processor (DQP)

Little needs to be said about the DQPs. These are the specific DBMS's capable of executing a DML command valid for that DBMS., i.e. ORACLE, INFORMIX etc.. They receive a valid data request, physically retrieve the data and return it to the requesting LQP.

We have examined the interfaces between each of the four query processors. Two application query scenarios are presented below to further illustrate how an application query is decomposed and handled at each level of processing.

II. Application Query Scenarios

An end-user interacts with the system by first selecting an application model to work with. This choice also implies a choice of a global schema since an application model can only refer to a single global schema. The end-user can then choose to execute any of the queries stored in that application model, or design and execute his own ad-hoc queries. In either case, the query will be sent to the AQP in the format specified earlier. As an example, assume the end-user chooses the student application model (Figure 4) and wishes to get information on all students majoring in physics and maintaining a certain grade point average. The message sent to the AQP would be:

```
(send-message 'AQP :query '((student (name major gpa)
                            ((= major "physics")
                             (> gpa 3.0))))
```

After performing the appropriate translation the AQP would then send the following message on to the GQP:

```
(send-message 'GQP :query '((gstudent (gname gmajor ggpa)
                            ((= gmajor "physics")
                             (> ggpa 3.0))))
```

The GQP then send messages to the Global Schema Manager to find out which tables contained data on the gname, gmajor and ggpa attributes. After finding out that gname is both gradetb and majortb, gmajor is in majortb (called concentration), and ggpa is in gradetb (called average) the GQP then asks the GSM how to join tuples in gradetb to those in majortb. The GSM would respond that joins could be done on the student name (gname). The GSM also provides the information that the LQP handling messages for gradetb was called GRADE, and that for majortb was called MAJOR. The GQP then sends the following messages to the MAJOR and GRADE LQPs:

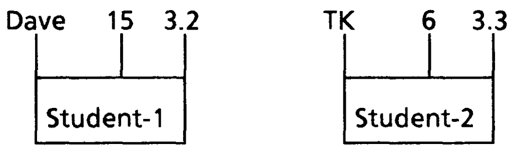
```
(send-message 'MAJOR :get-data '(majortb
                          (name concentration)
                       ((= concentration "physics"))))

(send-message 'GRADE :get-data '(gradetb
                          (sname average)
                       ((> average 3.0))))
```

The GQP then remaps the column names to the global schema nomenclature (i.e. sname --> gname) and performs a join on gname. The resulting table is then sent back to the AQP. The AQP remaps the table to the application model nomenclature (i.e. gname --> name) and instantiates application objects. For instance, after re-mapping in the AQP the results of the initial query might look like:

```
((name major gpa)
 (tk 6 3.3)
 (dave 15 3.2))
```

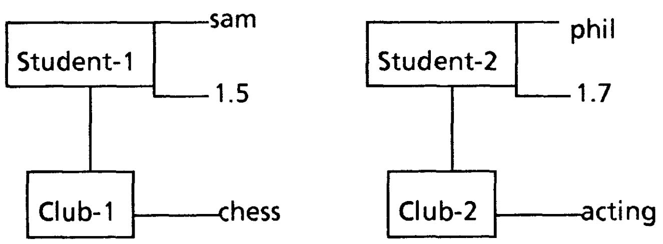
and the following two instance objects would be created:



A more complex query might be developed if a user were interested in looking at all students flunking out of the physics department and the clubs they belonged to see if there were any connection. This is a query that involves more than one entity in the application schema. The query sent to the AQP would be:

```
(send-message 'AQP :query '((student (name gpa)
                              ((= major "physics")
                              ( < gpa 2.0)))
                            (club (name)
                              ((> #members 3)))))
```

This query requests all the students in the physics department with a gpa below 2.0, and, for each of those students, the clubs which they belong to which have greater than 3 members. Let's look first at the final result the AQP should return. If two students, sam and phil, satisfied the criteria then the AQP should terminate by instantiating the following objects and relations:



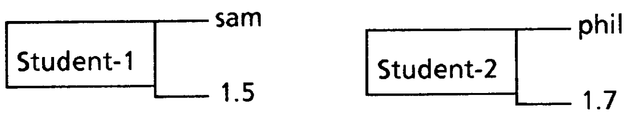
The AQP would proceed by first breaking down the multiple entity query into a set of single entity queries. The first message it would send to the GQP would be:

```
(send-message 'GQP :query '(gstudent (gname ggpa)
                           ((= gmajor "physics")
                           ( < ggpa 2.0))))
```

The GQP would proceed as before, translating and sending queries on their way to the appropriate LQPs and, eventually, DBMS. After the results were returned to the AQP and re-mapped to the application model nomenclature, the list might look like:

```
((gname ggpa) (sam 1.5) (phil 1.7))
```

The AQP would then instantiate the following two application objects:



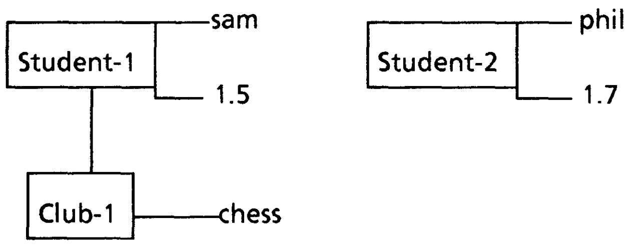
The AQP would then proceed to the second part of the query: finding the appropriate club for each of the student objects and linking them appropriately. One message would be sent to the GQP for each of the application object instances created earlier. First, the AQP inquires about sam's club affiliations:

```
(send-message 'GQP :query '(gclub (gname)
                           ((> #members 3)
                           ( = (student name) "sam"))))
```

Note that in order to send this message the AQP has only to strip off the section of the initial query referring to the club application object and add one condition which refers to one of the student objects already created. The GQP will process this message as before, eventually returning the re-mapped list:

((name) (chess))

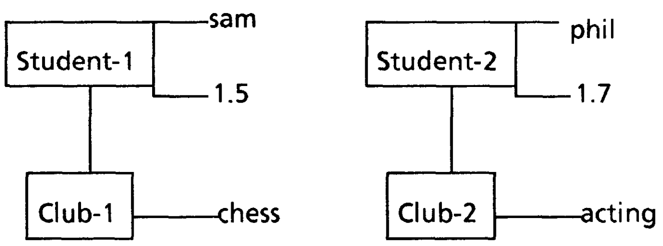
A club application object will then be instantiated and linked to the appropriate student object so that the application environment now looks like this:



Likewise, a message would be sent to the GQP concerning phil's clubs:

```
(send-message 'GQP :query '(gclub (gname)
                    ((> #members 3)
                    ( = (student name) "phil"))))
```

After re-mapping, instantiating and linking the results the AQP would have created the following objects, as initially desired:



The order of the application objects in the query that was initially presented to the AQP is significant. An inquiry about a group of students, and the clubs which they belong to, is quite different than an inquiry about a group of clubs and the students that are members. In the first case, a set of students is first selected, then the clubs which that set of students belong to *and* which meet the conditions placed on clubs are selected. There are thus two sets of conditions placed on the selection of clubs--indirectly, through the specification of the student members, and directly, through conditions about the club that are specified. In the reverse sequence the conditions placed on clubs are first applied to *all* clubs, then the students belonging to those clubs *and* satisfying any additional student conditions are selected. Thus, in this case, there is only one set of conditions placed on the selection of clubs.Thus, the arrangement of the objects in the query sent to the AQP is not interchangeable -- conditions applied to the first object will affect the selection of objects which they are later linked to. This is not thought to be a shortcoming since it is necessary to elicit this semantic information from the end-user before a query can be successfully processed.

III. A Closer Look at the GQP

Clearly, the GQP is the crux of the CIS/TK query processor subsystem. It has responsibilities for choosing where to retrieve the data from and how to join entities that are split between multiple tables. It is here that logical connectivity techniques such as attribute subsetting (a method of joining by employing heuristics) and credibility (table choosing) will be employed. The best way to understand the workings of the GQP is to follow up the earlier example in greater detail. That example only inferred the processes of the GQP by examining the syntax of the incoming and outgoing messages. Now we are in a position to examine those processes in detail.

Example: The GQP receives the following message from the AQP

```
(send-message 'GQP :query (gstudent (gname ggpa)
                                     ((= gmajor "physics")
                                     ( > ggpa 3.0))))
```

(1). The GQP parses the message, constructing a list of entities and attributes that are specified in the query:

```
    (setf eas ((gstudent gname) (gstudent ggpa) (gstudent gmajor)))
```

(2). The GQP sends a message to the GSM to find out which databases and tables will be needed:

```
    (send-message 'GSM :get-LQP-location '(eas))
```

The GSM returns the following list:

```
    ((gstudent gname ((STUDENTDB gradetb) (STUDENTDB majortb)))

    (gstudent ggpa ((STUDENTDB gradetb)))

    (gstudent gmajor ((STUDENTDB majortb))))
```

(3). The GQP examines the result, sees that two tables will have to be accessed (gradetb and majortb). It sends a message to the GSM in order to find out how to join these two tables:

```
(send-message 'GSM :get-LQP-join-key
'((STUDENTDB gradetb) (STUDENTDB majortb))
```

Assuming now that student id is the only join key (i.e. student name is coded very differently in the two tables), the GSM returns :

```
gid
```

(4). Now the GQP knows that in order to join entities from the two tables it needs the student id, so that data will be requested as well. In order to formulate the message for the LQP it must learn the column names of the desired attributes in the desired tables. The following messages are sent to the GSM:

```
(send-message 'GSM :get-LQP-nomenclature '(gstudent gid STUDENTDB
gradetb))

    --> id

(send-message 'GSM :get-LQP-nomenclature '(gstudent gid STUDENTDB
majortb))

    --> student-id
```

```
(send-message 'GSM :get-LQP-nomenclature
                    '(gstudent ggpa STUDENTDB gradetb))

   --> average

(send-message 'GSM :get-LQP-nomenclature
                    '(gstudent gname STUDENTDB gradetb))

   --> sname

(send-message 'GSM :get-LQP-nomenclature
                    '(gstudent gmajor STUDENTDB majortb))

   --> concentration
```

(5). Now the GQP is ready to send a query to each table which has a part of the information. First, the grade information is procured:

```
(send-message 'STUDENTDB :get-data '(gradetb (sname id average)
                                      ((> average 3.0))))
```

The LQP processes this query and sends an equivalent executable query to the appropriate machine and database. Assume that two people in gradetb meet the constraints. The STUDENTDB LQP returns:

```
   --> ((sname id average) (dave 011562530 3.2) (tk 034589221 3.5))
```

(6). The GQP re-maps the column names into the GS attribute names:

```
   --> ((gname gid ggpa) (dave 011562530 3.2) (tk 034589221 3.5))
```

(7). The query for majortb is sent , again to the STUDENTDB LQP:

```
(send-message 'STUDENTDB :get-data '(majortb (student-id)
                                         ((= concentration "physics))))
```

The STUDENTDB LQP returns:

```
--> ((student-id ) ((011562530) (234847729) (002348614))
```

(8). The GQP re-maps student-id into the GS attribute name:

```
--> ((gid) ((011562530) (234847729) (002348614))
```

(9). A join is performed on gid using the two re-mapped lists, and only the originally requested information is retained:

```
--> ((gname ggpa) (dave 3.2))
```

(10). Finally, this result is returned as the response to the AQP's initial message.

Mapping Data Between Different Representation Schemes in CIS/TK

Part I of this report detailed the basic design of the CIS/TK system, using several simple queries to illustrate the design approach. That approach was driven by a desire to facilitate several aspects of logical connectivity that we can now address more directly. In particular, this report will examine how the CIS/TK system architecture is enhanced to deal with issues such as mapping data between different representation schemes. A Placement Assistance System (PAS) will be used to generate illustrative examples. Our goal is to incrementally provide real, exciting, and non-trivial examples.

Knowledge, of course, can be communicated in many different fashions. Accordingly, we typically find the same type of information represented in many different ways in a distributed environment. The PAS application model in Figure 1 provides us with an example. As shown in this figure, students have relationships with the types of positions that they may apply for (i.e. course 15 students may apply for management positions, course 6 students for engineering positions etc.). Each position, in turn, is related to the salary structure information for that industry, and for the city in which that position is located. A student entity in the PAS model also has a one-to-many relationship with a prior degree entity, which contains grade information for that student's previous degree courses. This grade information might be recorded in a variety of fashions: a five point scale, a four point scale or a letter grade scale. However, an application end-user will generally prefer to view all information in a consistent representation. For example, officers in the MIT graduate admissions office might prefer to view all applicant's transcripts using a five point grade scale since this is what MIT uses and they wish to judge all applicants in a consistent

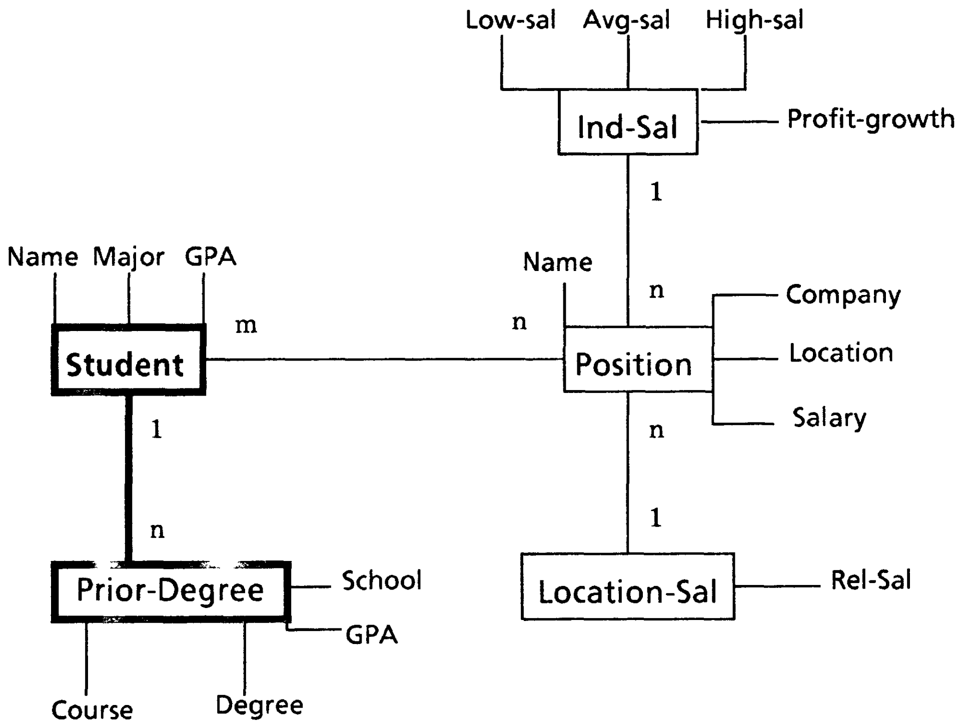


Figure 1 A PAS APPLICATION MODEL

fashion. Furthermore, such consistency will be needed to join entities residing in different tables. For example, students might be identified by an employee id in one database and a student id in a second database. Combining information from both sources requires translating from the one value representation scheme (or domain) to the other. Providing this consistency in a flexible, user-driven environment is a primary goal of our CIS system.

I. Storing Translation Knowledge

There are three types of information that must be captured by the system for any information that can be represented by more than one domain:

(1). What are the possible set of domains for this information (i.e. 5-pt 4-pt and letter for grade) ?

(2). Which of these domains is utilized by each of the local DB's which contain data for this attribute ?

(3). How are values from one domain translated to another ?

If these three pieces of information are known, then any attribute can be translated from any one domain to another in order to either satisfy the end-user's desires or to automatically prepare for joins. The method of translating from one domain to another (the answer to (3), above) may be defined in any of three ways: by a procedural method, an Inter-Database Table (IDT) lookup, or heuristics. A procedural method is a translation which is accomplished by simply substituting the current domain value into a previously-defined function. An example is the translation from a 4 point scale to a 5 point scale: it is done by simply substituting the 4 point value (i.e. 3.3) into the equation 5-pt = 4-pt * 1.25 (result : 4.125). An IDT lookup is simply a table lookup request where the lookup key is the current domain value, i.e. translating from a letter grade to a 5 point scale is easiest done by providing a table of letter grades and their corresponding values in the 5 point scale, as shown in Figure 2. Finally, heuristics is the method designed for less well-structured translations. While the first two methods required only the current domain value to perform the translation, some translations require additional information about associated attribute values. As an example, consider again the PAS application model in Figure 1. A student in that model has a relationship to all the possible positions that he may interview for. One of the slots in the position object is, of course, salary.

BOOTDB

Letter	4-pt.	5-pt.
A	4.0	5.0
B	3.0	4.0
C	2.0	3.0
D	1.0	2.0
F	0.0	1.0

Figure 2 Grade Inter-Database Table (IDT)

A simplified global schema for the application model is shown in Figure 3. The local STUDENTDB and DEGREEDB databases are exemplified in Figure 4a, and PASDB in Figure 4b. Salary data can be represented in one of three domains: dollars/yr (i.e. 48,000), dollars/month (i.e. 4,000) or competitive-description (i.e. less competitive, competitive, highly competitive). Translating from the competitive-description domain to the dollars domain requires additional information: location, industry, etc.. This is because "competitive" means one thing for a consulting position in New York and something different for a marketing position in the Midwest. Thus, the heuristics method employs rules and a broader set of data to translate between domains, as compared to procedures and IDT lookups. We believe that these three methods (procedures, IDT lookups and heuristics) provide a robust manner of mapping values from one domain to another. The knowledge of how to translate from one domain to another is stored in an object hierarchy,below the translate class, which is depicted in Figure 5. After examining how these translate objects function, the application model and global schema objects will be re-examined to see what extensions need be made in order to represent the necessary domain knowledge.

Finally, two detailed scenarios will be presented to see how translation is accomplished in both simple and complex cases.

The translate object is a subtype of the CLASS class, the top-level object class in KOREL. The translate object contains a method called :translate which accepts a request for a translation from one domain to another. The translate object also has slots called procedures, IDT-lookups, and heuristics. These slots are not given values at this level.

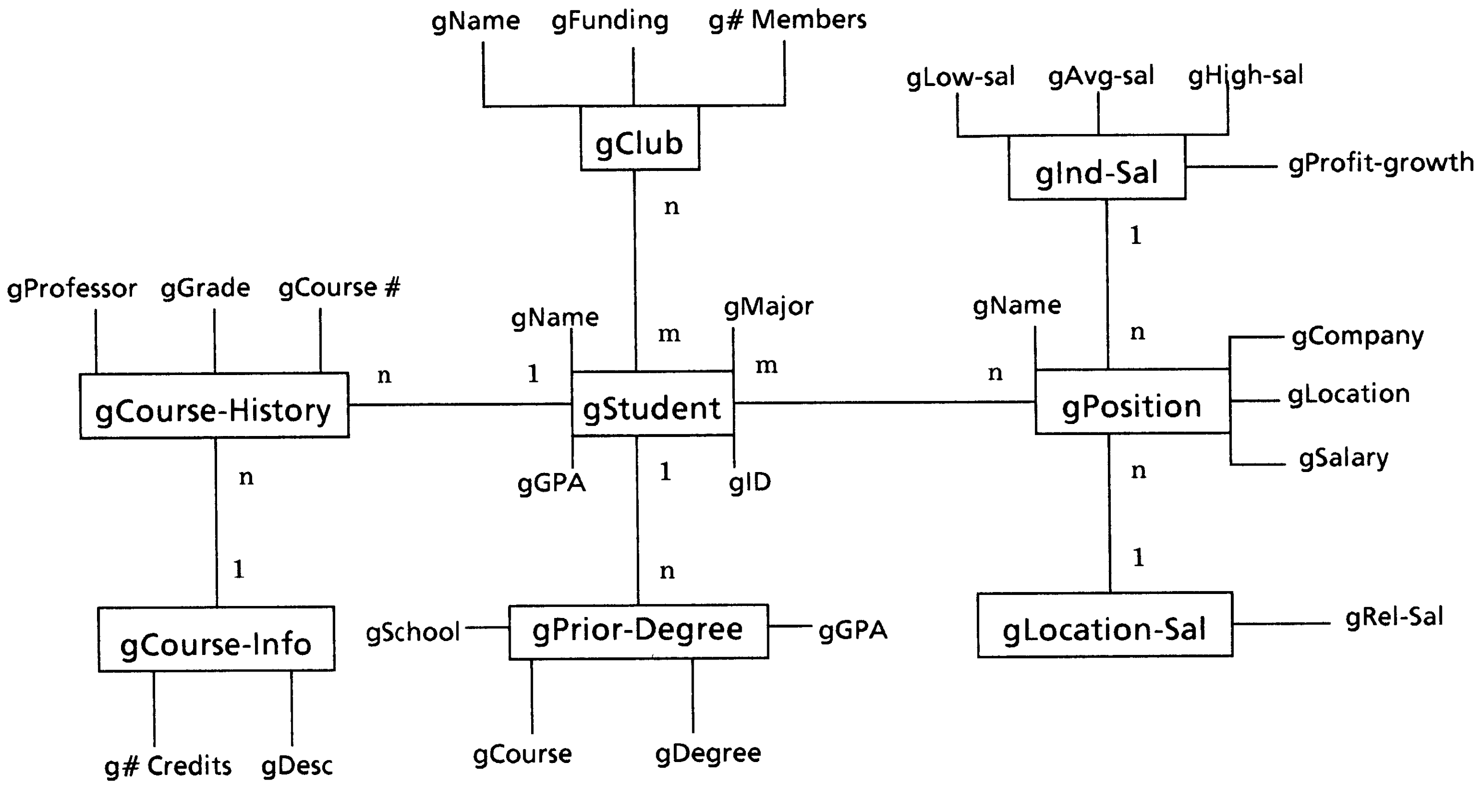


Figure 3 A SIMPLIFIED GLOBAL SCHEMA FOR PAS

There are subtypes of the translate class for each attribute which is represented by more than one domain in the application model. For example, Figure 5 shows

STUDENTDB

Grade tb

Sname	sID	Average
Dave	011562530	4.5
TK	013216309	2.0
Phil	029786133	5.0
.	.	.
.	.	.
.	.	.

Major tb

Sname	ID	Concentration
Wong	011562530	6
Horton	011562530	15
Jones	029786133	6
.	.	.
.	.	.
.	.	.

DEGREEDB

Degree tb

ID	School	Course	Degree	GPA
011562530	Amherst	Economics	B.A.	3.3
056250023	U. Texas	Physics	B.S.	4.0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Figure 4a THE STUDENTDB AND DEGREEDB IN THE LOCAL DATABASES

grade-tr and salary-tr as two subtypes of the translate class. Finally, there are instances of each of these sub-classes which represent each of the possible domains. For example, the grade-tr class has 4-pt, 5-pt and letter instances. These instances also are given values for the procedures, IDT-lookups and heuristics slots. The

PASDB

Positiontb

Company	Position	Salary	Location	Indcode	Course
IBM	Salesperson	Competitive	N.Y.	42	15
Microsoft	Program Manager	Highly competitive	Seattle	42	6
Microsoft	Product Manager	Highly competitive	Seattle	42	15
Ford	Product Planner	Null	Detroit	41	15
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

Salarytb

Indcode	Org-salary	High-range	Low-range
42	38,000	48,00	34,000
41	42,000	50,000	38,000
.	.	.	.
.	.	.	.
.	.	.	.

Industrytb

Indcode	Profitgrowth
42	1.05
41	1.15
.	.
.	.
.	.

Locationtb

City	Relative pay
N.Y.	1.25
Seattle	.96
Seattle	.82
Chicago	1.05
.	.
.	.
.	.

Figure 4b THE PASDB IN THE LOCAL DATABASES

values of these slots for the 4-pt, 5-pt and letter instances are shown in the Figure below.

OBJECT	SLOT	VALUE
4-pt	instance-of	(grade-tr)
	procedures	((5-pt chg4-to-5))
	IDT-lookups	NIL
	heuristics	((letter 4-to-letter-rules)
5-pt	instance-of	(grade-tr)
	procedures	((4-pt chg5-to-4))
	IDT-lookups	NIL
	heuristics	((letter 5-to-letter-rules))
letter	instance-of	(grade-mk)
	procedures	()
	IDT-lookups	((4-pt BOOTDB letter-IDT let__scale 4__scale) (5-pt BOOTDB letter__IDT let__scale 5__scale))
	heuristics	NIL

Each instance object (i.e 4-pt, 5-pt, letter) contains the knowledge for translating from that domain to any other. If the mode of translation to a given domain is a procedure, the applicable procedure is stored in the procedure slot along with the domain name as an identifier. If the method of translation is an IDT-lookup then the IDT database, table, and colum names are stored in the in the IDT-lookups slot with the domain name. If the translation requires heuristics then the rule set identifier is stored in the heuristics slot along with the domain name as an identifier. Thus, as

OBJECT	SLOT	VALUE
salary-tr	superiors	(translate)
	instances	(ann-sal mon-sal comp-sal)
ann-sal	instance-of	(salary-tr)
	procedures	((mon-sal ann-to-mon))
	IDT-lookups	()
	heuristics	((comp-descript ann-to-comp-rules))
mon-sal	instance-of	(salary-tr)
	procedures	((ann-sal mon-to-ann))
	IDT-lookups	()
	heuristics	((comp-descript mon-to-comp-rules))
comp-descript		
	instance-of	(salary-tk)
	procedures	()
	IDT-lookups	()
	heuristics	((ann-sal comp-to-ann-rules) (mon-sal comp-to-mon-rules))

the Figure above shows, translating from 4-pt to 5-pt requires a procedure while translating from letter to 4-pt is done by an IDT lookup. The translate method, defined in the superior object class translate, performs the actual translation. It allows any of the instance objects to accept a message of the form:

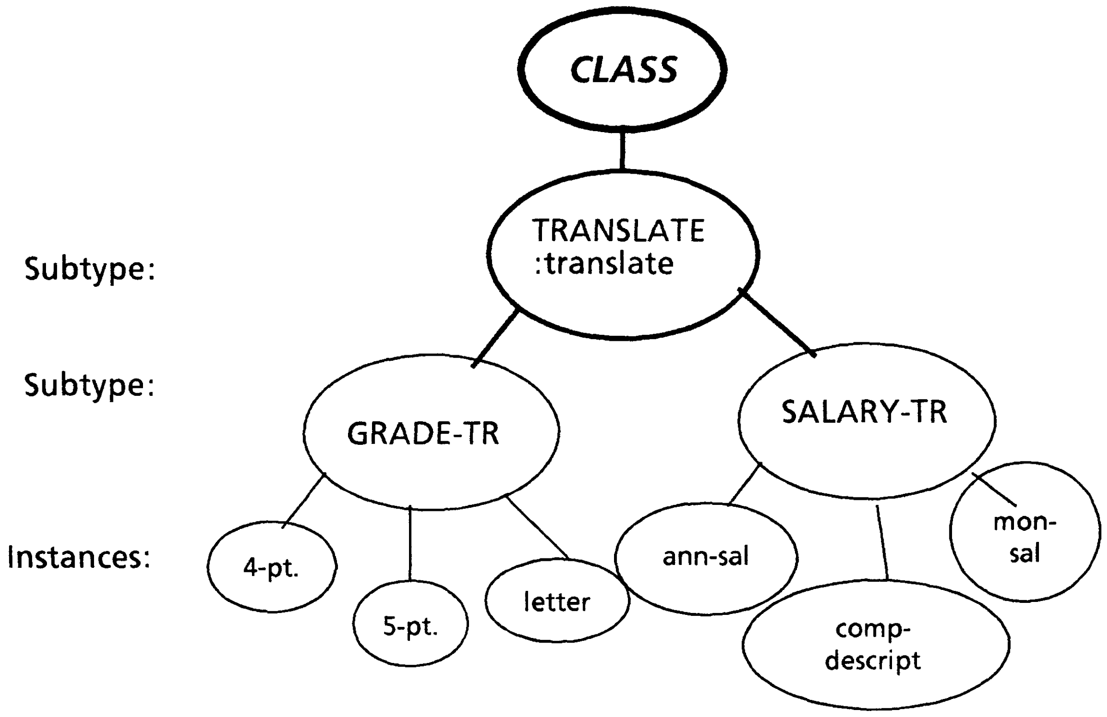


Figure 5 THE TRANSLATE OBJECT HIERARCHY

```
(send-message 'rep-from :translate '(value rep-to))

        i.e.
          (send-message '4-pt :translate '(3.3 5-pt))
```

which, in this case, would return the value "4.125". The translate method works as follows: It first examines the procedures, IDT-lookups and heuristics slots to see where the knowledge of translating to the specified domain is kept. It is assumed that there is only one encoded method of translating from one domain to another, thus the order of search is not significant. If the appropriate knowledge is found in the procedures slot then the associated procedure is called with a single argument--the value which was passed to the translate method. For example, the translate message above would result in the following procedure call:

OBJECT	SLOT	VALUE
translate	superiors	(CLASS)
	subtypes	(grade-tr salary-tr)
	procedures	()
	IDT-lookups	()
	heuristics	()
	methods	(:translate trans)
grade-tr	superiors	(translate)
	instances	(4-pt 5-pt letter)

```
(chg4-to-5 3.3)
```

the value of which is then returned as the response to the initial translate request message.

If the knowledge is found in the IDT-lookup slot then the translate method generates the appropriate table lookup request by preparing the following message:

```
(send-message 'LQP :get-data '(table (rep-to)((= rep-from value))))
```

Note that the appropriate LQP, table and column names are known since they are stored in the IDT-lookups slot of the instance object. For example, if the translate method were requested to translate from a letter grade to a 4-point grade, i.e.

```
(send-message 'letter :translate '("B", 5-pt)
```

Then the translate method would generate the following table lookup request:

```
(send-message 'BOOTDB :get-data '(grade-IDT (5__scale((= let__scale "B"))))
```

which, in turn, would return the following table:

```
((5__scale) (4.0))
```

The translate method then strips the extraneous information from the list and simply returns the value: "4.0" in response to the initial message.

If heuristics are needed to translate from one domain to another, as in the case of salary (from competitive-description to dollars/yr), then things are a bit more complicated because of the need to procure additional information. Frequently, that information will need to be retrieved from the database since it need not have been directly specified by the end-user query. For instance, the end-user may be interested in a particular position and its salary, but not its location. The location for that position will have to be retrieved, however, in order to translate the salary information from "competitive" to a dollar amount since the rules for performing the translation (Figure 6) use location to help infer a dollar value for salary. Note that this database retrieval is quite a bit more complicated than in the IDT case: here there is no way of identifying ahead of time where the appropriate information will be found--it depends on the particular position identified by the end-user query.

For this reason, translations requiring heuristics are performed at a higher level than those utilizing simple procedures or IDT lookups. The former are done by the

AQP, while the latter are performed by the GQP. The preference is to try to perform joins at as low a level as possible, in order to decouple the knowledge of physical data location from the conceptual orientation of the AQP. Furthermore, the GQP was designed to accept requests dealing with only a single global schema entity in order to simplify the information exchange between itself and the AQP and to more clearly define its role. Since heuristics may well require knowledge about diverse entities, these translations and joins are performed at the AQP level.

A powerful feature of this design is that domains may be defined by the application builder, as well as by a database designer. For example, there may be only a single database which contains salary information for positions, where salary is recorded by the competitive-description domain ("less competitive", "competitive", "highly competitive"). However, the application builder may well decide that it's valuable to view salary information in dollars per year as well. By defining this domain, and the knowledge of how to translate between competitive-descriptive and dollars per year (in this case that means heuristics), application users are presented with two possible representations of salary information even though the databases only contain the one. A more detailed look a precisely how heuristic translations are accomplished will be presented later in this chapter.

II. Extensions to GS entities and AM objects

Now that we have established how the translate objects accept and act upon translation requests, it is appropriate to investigate just how these requests are generated. Previously, GS entities were defined to contain a mapping of attributes to database, table and column name location. Now this implementation will be further extended in order to indicate how the domain representation for each of these databases as well. As an example, the gposition entity is implemented as shown in the Figure below. Note that the only change is that a domain identifier has been

OBJECT	SLOT	VALUE
gposition	gsalary	((PASDB postitiontb salary comp-descript))
	gname	((PASDB positiontb position NIL))
	glocation	((PASDB positiontb location NIL))
	.	
	.	

added to the Database/Table/Colum-name information. Since there are three ways of representing salary (comp-descrip, ann-sal, mon-sal) this kind of information is needed. On the other hand, both name and location are only represented in one way, thus there is no need to represent any translation knowledge and the domain is specified as NIL.

The AM objects must be extended as well, in order to record the possible domains that the end-user can choose from. This is done by simply including the specific translate class object in the object slots, as below:

The salary slot now contains a symbolic pointer to the position-tr class object. This allows the user to request a listing of possible domains for salary, which can then be provided by listing the instances of the position-tr object. Again, since name and location are only available in one domain there is no information about their translation class objects recorded in the AM position object.

OBJECT	SLOT	VALUE
position	salary	(pas-gschema gposition position-tr)
	name	(pas-schema gname)
	location	(pas-schema glocation)
	.	
	.	

III. Two detailed scenarios

Let's first examine a fairly straight-forward case of mapping between domains. Assume that the end-user wants to see the names and grade-point averages of all students above a certain level. The only catch is that he wants to view grade averages on a 4-point scale, not the 5-point scale by which they are actually stored in the database. (The applicable global schema is shown in Figure 4 and the local schemas in Figure 4a). His query will result in the following message being sent to the GQP:

```
(1)     (send-message 'GQP :query '(gStudent (gName (gGpa 4-pt))
                                    ((> gGpa 3.5))))
```

Note that the query format has been slightly altered: now that we deal with multiple domains we must specify, for any attribute which has more than one domain representation, which of those domains is desired. In this case the desired domain, 4-

pt, is now specified in the message as well as the desired attribute itself, gGpa. Note also that the value on the condition for gGpa is also specified in terms of the desired 4-pt scale. This consistency is important: if the user wishes to view grades on a 4-point basis then he should also be able to discuss any conditions he applies to grades in the 4-point domain language.

(2). As before, the GQP must choose which databases and tables to get the data from. In this case, data for both gName and gGpa can be retrieved from the gradetb table in StudentDB. However, the grade information is in the 5-pt scale. This fact is determined by the Global Schema Manager (GSM) by means of a new method called :get-LQP-domain. Thus the condition on gGpa will have to be translated from the 4-pt to the 5-pt scale before constructing the Abstract Local Queries. Accordingly, the GQP sends the message:

```
(send-message '4-pt :translate '(3.5 5-pt))
```

which, in turn , results in the following procedure call:

```
(chg4-to-5 3.5)
```

which returns a value of 4.37 . At this stage the GQP also constructs for itself a list of domain mapping requests that it will need to perform after the data is retrieved. In this case, there is only one such mapping:

```
((gGpa (5-pt 4-pt)))
```

(3). Now the GQP is ready to send the Abstract Local Query to the StudentDB LQP:

```
(send-message 'StudentDB :get-data '(gradetb (sname average)
                                       ((> average 4.37))))
```

returns the following list:

```
-> ((sname average) (Dave 4.5) (Phil 5.0))
```

(4). As before, the column headings are mapped back into the GS attribute names:

```
--> ((gName gGpa) (Dave 4.5) (Phil 5.0))
```

(5). The next order of business is to perform the domain mapping request for gGpa from the 5-pt scale to the 4-pt scale. The GQP sends the following message:

```
(send-message '5-pt :translate '(4.5 4-pt))
```

which, in turn, causes the following procedure to be called:

```
(chg5-to-4  4.5)
```

and the value 3.6 to be returned. Next, the same process is used to map Phil's grade from a 5-point to a 4-point scale:

```
(send-message '5-pt :translate '(5.0 4-pt))
```

which returns the value 4.0 .

(6). Finally, these values are replaced in the list. The GQP has now performed the task requested of it.

```
(gName gGpa) (Dave 3.6) (Phil 4.0))
```

This was an example of a translation which required knowledge that was represented in procedures (chg4-to-5 and chg5-to-4). If the translation knowledge were embedded in a table (IDT), the process would have been identical except that the :translate method would have generated a table lookup request rather than a procedure call.

Now let's examine the more complex case of how a translation requiring heuristics would be handled. Assume that the end-user wished to see all the potential positions in New York and their salaries, in terms of dollars per year. Figure 1 shows the Application Model, Figure 3 the applicable Global Schema and Figure 4b the local databases. The following message would be sent to the AQP:

(1).
```
(send-message 'AQP :query '(position(company-name pos-name salary
                                    ((= location "N.Y."))))
```

(2). The query is translated into the necessary GS query:

```
(send-message 'GQP :query '(gposition(gcomp-name gpos-name (gsalary
ann-sal))((= glocation "N.Y"))))
```

Note that the requested domain for salary is now included in the query message to the GQP. The GQP will attempt to return the salary information to the AQP in this format if possible.

(3). The GQP performs the following tasks:

A. Find out where data for the attributes is stored.

B. Choose data sources, if possible using a DB which has the data in the desired domain.

C. As in the previous case, if the domain for any attribute in a condition is specified differently than in the underlying DB, then try to translate the condition. This time, there are no such cases.

D. Construct a list of values to be translated upon return (in this case only salary is included) :

```
        ((gsalary (comp-descript ann-sal))
```

(4). Send out the Generic Local Queries to the LQPs:

```
        (send-message 'PASDB :get-data '(positiontb (company position
        salary)((= location "N.Y.")))
    returns:
      -->   ((company position salary)(IBM salesperson competitive))
```

(5). Re-map column names to GS attribute names:

```
      --> ((gcomp-name gpos-name gsalary)(IBM salesperson competitive))
```

(6). Try to fulfill all the translation requests--only procedures and IDT lookups can be used at the GQP level, thus the gsalary mapping request fails.

(7). Re-map GS attribute names to AM names, marking the unsatisfied translations by including the current domain.

Returned to AQP:

```
-->((company name(salary comp-descript))(IBM salesperson competitive))
```

(8). The AQP instantiates the position objects from the returned list. The query is done if no translation requests are outstanding.

(9). Since salary still must be translated (from comp-descript to ann-sal), the AQP gets a list of the information necessary to perform the translation from the translation object hierarchy:

```
        (send-message 'comp-descript :info-needed '(ann-sal))
    --> ((position salary)(industry-salary average)(location-salary rel-salary)
                                        (industry-salary profit-growth))
```

This is accomplished by reading the rules, as shown in Figure 6, for the translation and finding out which pieces of information is needed

(10). Now the AQP must generate queries to procure the needed information. The trick is to somehow ensure that we get the industry and location salary data that is related to the specified position. This is done by including the previous condition (i.e. "located in N.Y.") plus a condition for each slot value in the instance. Thus we essentially re-specify the position, and then take the industry and salary information which is linked to that position. (If there are multiple instances proceed one at a time).

```
(send-message 'GQP :query '(gind-sal(gaverage gprof-growth)((= glocation "N.Y")
        (= (gposition company) "IBM")(= (gposition gpos-name) "salesperson")
        (= (gposition salary descriptive)"competitive"))))
```

```
(IF ( = (position  salary) "less competitive")
(THEN ((position salary)  =  (Ind-Sal low-sal) * (Ind-Sal profit-growth)
                                   * (Location-Sal Rel-Sal)))

(IF ( = (position  salary) "competitive")
(THEN ((position salary)  =  (Ind-Sal avg-sal) * (Ind-Sal profit-growth)
                                   * (Location-Sal Rel-Sal)))

(IF ( = (position  salary) "highly competitive")
(THEN ((position salary)  =  (Ind-Sal high-sal) *
                                   (Ind-Sal profit-growth) *
                                   (Location-Sal Rel-Sal)))
```

Figure 6: Heuristics for translating salary from comp-descript to ann-sal

```
--> ((gaverage gprof-growth)(42,000 1.12))
```

(11). Instantiate the industry-salary object, and repeat the previous step for location-salary object. After instantiation, we have values for average-sal, profit-growth and rel-sal.

(12). Now we can translate from "competitive" to dollars per year.

```
(send-message 'descriptive :translate '("competitive", ann-sal))
```

(13). Translate finds the knowledge for translating to ann-sal in the heuristics slot, thus performs the following sequence of actions:

```
(take-knowledge pas-model)
(grab-rules descriptive-to-$)
(apply-rules)
```

(14). The result is that the value of the salary slot is set to:

salary = 42,000 * 1.05 * 1.12 = 49,392

Attribute Subsetting in CIS/TK

The ability to join information about a particular entity from disparate databases is clearly a necessary feature of a CIS system. When possible, these joins are performed using a primary-foreign key relationship. Sometimes, however, this type of join isn't possible, either because no common unique key exists (e.g. students coded by id in DB #1 and name in DB #2) or because a common unique key does exist, but is coded ambiguously (e.g. a corporation coded as "IBM Inc." in DB #1 and "IBM Corp." in DB #2). In such cases a technique called attribute subsetting, employing heuristics, may be used to join entities from the two databases and, optionally, retain this mapping in an Inter-Database Instance Identification Table (IDIIT) for later use (see Figure 1 for an example IDIIT for corporation entities). This report will detail the concepts involved in attribute subsetting and lay out a rough design for implementing this functionality in the framework of the CIS/TK system.

CompanyDB	ReutersDB
IBM Inc.	IBM Corp.
British Petroleum	BP Inc.
Ford Motor Co.	Ford

Figure 1. An IDIIT for Two Company LQPs

I. An Example of Attribute Subsetting

Imagine a professor and his teaching assistant (TA) discussing the performance of one of their students. We can view each of them as maintaining a database containing various types of information on the same group of students. A conversation will typically begin by the professor identifying one of the students by name, following which both will volunteer information about that student (i.e. grades, performance, etc.). This is an example of joining information from two tables by means of a primary-foreign key join--in this case, using student name as that key. Now assume, however, that while the professor knows the students by name, the TA identifies them by means of nicknames that he has attached to them (i.e. Sleepy, Dopey etc.). Now they face a real problem of making sure that they are even talking about the same person since there the mapping between names and nicknames isn't captured--there is no longer a primary-foreign key relationship. However, they are likely to pursue other ways of mutually identifying the student, as per the following discussion:

(Professor):	Do you know who TK Wong is ?
(TA):	No. Does he come to the morning class ?
(Professor):	Yes, when he comes at all.
(TA):	How well is he doing in the class ?
(Professor):	Not well. He's always falling asleep.
(TA):	Is he quiet ?
(Professor):	No ! He keeps complaining about our LISP compiler.
(TA):	Oh, sure ! I call that guy Big Mouth.

So, even though there is no common unique key, there may be a way of using other shared (non-unique) attributes (i.e. attendance, performance etc.) which can be used to eliminate all other possibilities. This approach we call attribute subsetting. At first glance, this may seem like nothing more than searching for a common multiple-key unique attribut. There are two reasons why attribute subsetting is different. First, there may be no way to eliminate all the possibilities: the professor and TA may, for instance, at best reduce the possibilities to three. At that point they may pick up the student directory and look at pictures of the three students for final identification. However, this will be much less work than checking through the pictures of each class member. Thus, in a database environment, while attribute subsetting can help identify entities some (hopefully small) degree of user interaction will be required as well.

The second reason is more interesting. As well as comparing shared attributes, the professor and TA may also be able to infer values for other attributes which can then be used in the subsetting process. As an example, consider the same type of professor-TA example as above, this time as pictured as in Figure 2. Now the TA is trying to identify someone the professor calls Jane Murphy. In this case, there is only two shared attributes: attendance and performance. Comparing values for these two attributes eliminates only two possibilities (Dopey and Dreamer) still leaving five other possibilities. However, there are relationships between the other attributes which allow the subsetting process to continue. For instance, a person's age implies something about their grad/undergraduate status. Thus, even though status isn't stored in the professor's database he is able to infer a value for it since he *does* have age information. This allows the TA to eliminate the student he calls Whiner from consideration, since Whiner is known to be a graduate student while Jane's age infers that she is an undergraduate. Following along, we note that name frequently implies a value for sex, which allows the TA to eliminate all the males from consideration (Shifty). The student's address implies something about how they are transported to school--the fact that Jane Murphy lives in Marblehead, greater than 20 miles from school, means that walking and biking are unlikely modes of transportation for her. This allows the TA to eliminate still more candidates (Nerdy). Finally, the fact that Jane Murphy is registered in two MIS

Q: How does Dave evaluate Jane?

Database #1 (Rich, Instructor for 564 and 579)

Name*	564	579	Sec564	Age	Perform	Address
Jane Murphy	Yes	Yes	A.M.	19	Strong	Marblehead

Database #2 (Dave, head TA for 564)

Nkname*	Sec564	Perform	Sex	Maj	Status	Trans	Evaluation
Excellency	A.M.	Strong	F	MIS	UG	car	sharp cookie
Saavy	A.M.	Strong	F	Mgt	UG	train	Coordinator
Nerdy	A.M.	Strong	F	MIS	UG	bike	hacker
Shifty	A.M.	Strong	M	MIS	UG	car	wild card
Whiner	A.M.	Strong	F	MIS	G	car	tough cookie
Dreamer	A.M.	Weak	M	?	?	?	discard
Dopey	P.M.	Good	M	MIS	G	walk	routine worker

- *564 section & performance* only common attributes
- Reduces possibilities from *entire database* to 5.
- Other relationships in the data to take advantage
  - age -- status (UG)
  - name -- sex (F)
  - location -- transport (not bike)
  - course load -- major (MIS)

Figure 2: ATTRIBUTE SUBSETTING

courses means that it is likely that she is an MIS concentrator. This allows the TA to reduce the candidate set to one--Excellency.

Thus, even though only a few attributes are common to both databases, further comparisons can be made because of the relationships between the data. These kinds of relationships are likely to occur in a CIS system precisely because of the heterogeneity: fragmentation of information is frequently caused by the fact that separate organizations are interested in different attributes of the same entity. For example, the registrar's office is likely to be concerned about a student's course schedule and home address, the bursar's office is likely to be concerned about his financial status (tuition and fines owed) while the campus police track whether he has been issued a parking sticker. In such a system there may be little opportunity to directly compare data between the different databases. Using heuristics, though, may allow us to make further comparisons. For instance, whether or not a person has a parking sticker implies whether he owes a parking fee to the bursar's office--if he doesn't have a sticker then there can be no fee attributed to him at the bursar's office.

II. Attribute Subsetting in CIS/TK

Attribute subsetting fits in with the CIS/TK system in the following way. When a Global Schema (GS) query is processed which requires joining information about a single entity from different databases, a primary-foreign key join will first be attempted . If no join key is found then the IDIIT for that entity is searched (if it exists). If it doesn't exist, or no entry is found for that entity, then attribute subsetting is performed. The selected entity in the one table must be compared against all the entities in the other table. Ideally, all possibilities save one will be eliminated. If so, this entity will be presented to the user for confirmation and, if confirmed, will be entered in the IDIIT. If more than one possibility exists the user will choose between them. If no possibilities exist after all comparisons, the problem is either a bad piece of data somewhere, or a heuristic that doesn't cover a special case (i.e. a very young age usually, but not always, implies that the student is an undergraduate). In this case the system will begin backtracking, presenting the list of possibilities to the user in order of likelihood.

It is clear that the implementation of the subsetting process will require a few different capabilities. First of all, the system must support heuristics--there must exist a way to represent the relationships between attributes. Secondly, there must be a way to represent the degree of uncertainty that we associate with attribute values since we will be using the heuristics to infer less-than-certain values for attributes. Finally, in the case where there are no perfect matches, there must be a way of determining a ranking of possible matches so that the user can select a match as efficiently as possible.

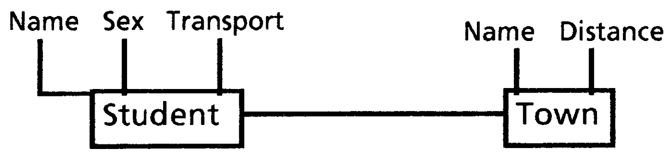


Figure 1

Heuristics, of course, will be represented by rules. The idea is that these rules will define the set of possible values for one attribute, given a known set of values for some other attributes. For instance, in the example above, a student's address (Marblehead) determines the type of transportation he uses to get to school. Let's examine just how this happens. Figure 3 shows a global schema for such a student. At the local level, there is a database of student information as well as a geographical database of distances between towns. Thus, the student object is joined to the town object, which has an attribute called distance, the value of which is the distance between his home and Cambridge. However, through the transport slot, the student object is also related to the mode-of-transportation object, which is listed below. The mode-of-transportation class object has three instances: bike, car and train. Note that these objects all display a new facet called DOMAIN. The DOMAIN facet holds the information about which domain a slot can draw values from. The information about domains is kept in the DOMAIN object, which is also shown below.

Mode-Of-transport Class Object

Slot	Facet	Value
Type	VALUE	?
	DOMAIN	dCarriers
Distance-covers	VALUE	?
	DOMAIN	dRange
Fuel	VALUE	?
	DOMAIN	dFuel

Mode-of-Transport Instances

Bike	Car	Train
NEAR	NEAR, FAR	FAR
Muscle	Gas	Oil,Coal

Domain Object

Slot	Facet	Value
dCarriers	VALUE	?
	CHOICES	(Bike, Car, Train)
dRange	VALUE	?
	CHOICES	(NEAR, FAR)
dFuel	VALUE	?
	CHOICES	(Muscle, Oil, Gas, Coal)

The student is linked to the mode-of-transportation object through the DEFAULT facet of the transport slot. Thus, if no data on a student's transportation is retrieved from the database, he is still linked to a set of choices beneath the mode-of-transport object. Heuristics will be used to try to restrict his possibilities even further.

Student Class Object

Slot	Facet	Value
Name	VALUE	?
Sex	VALUE	?
Transport	VALUE	?
	DEFAULT	mode-of-transport

Heuristic

```
(IF ((town distance) > 20)
   (THEN (domain dRange) = "FAR")
   (ELSE (domain dRange) = "NEAR")
```

The subsetting process works in the following manner. After data has been retrieved from the database, a student object is instantiated (i.e. "dave", "male", "Marblehead"). Next, the heuristic rule set is run. Heuristics can either set a value for a slot in the domain object, as above, or they can directly set the VALUE or

CHOICES slot in a GS entity. In the case above, the VALUE facet of the dRange slot in the the Domain object is set to "FAR". After running the rules, we try to restrict the values for any slots in the student instance which has no value in the VALUE facet, i.e. Transport. From the DEFAULT facet of the Transport slot, we see that the possible choices are the mode-of-transport instances. Because of the DOMAIN facet in the mode-of-transport objects, we are able to see that we can further restrict the choices based on the fact that the dRange domain is one determinant of mode-of-transport. Since the dRange value was set to "FAR", only car and train are possible modes of transportation. Thus, the CHOICES slot of the transport object is set to the value ("car", "train").

The power of this approach is that the rules are allowed to be quite general and intuitive in scope. By avoiding having a rule for every town we make the rule set smaller and much more accessible to the user. It also allows us the possibility of leveraging several inferences with a single rule: for instance, the time a student sets his alarm clock for may also be a function of how close to school he lives. No new rules would be needed to implement this since the distance concept has already been established in the domain object.

The harder question arises if, after attribute subsetting, none of the candidates match on all slot values: then we would like to present them to the user in an order starting with those most likely to be the match. If we can accomplish this we will be able to converse knowledgeably with the user. Doing this, however, implies some knowledge of which attribute value mismatches are more significant. Understanding this, in turn, requires understanding how attribute values for the same entity can be recorded differently in different databases. There are four reasons why this might be so:

(1). Data entered incorrectly.
(2). Data values change over time.
(3). Same data viewed differently by different database designers.
(4). Values for data only inferred, not certain.

We will not examine the first case in detail, since this type of data error is idiosyncratic and generally occurs randomly across databases and attributes. However, the framework that we present to handle cases 2-4 can be extended to handle this type of data error if more is known about the specific error rates of different databases.

An example of the second type of problem would be using a person's age to help identify him in two different databases: age changes over time, thus, if the databases are updated at different intervals, we may have the same person represented by two different ages in two different databases. Thus, because of this, age is a poorer measure by which to identify a person than, for example, birthdate which doesn't change over time. Hence, we are less concerned about a mismatch on age than a mismatch on birthdate--if in trying to find a match we were at best able to find two possibilities which had only one mismatch each, one on age, one on birthdate, then we would present the one that mismatched on age first.

An example of the third type of problem above is exemplified in using street address to identify a university: two different people might view the meaning of street address differently. A defense contractor views MIT's street address as the address of the defense liason office, while the federal loan office views MIT's street address as the address of the bursar's office. Thus, street address would be a poor attribute with which to try to identify university entities, as opposed to, for example, state location, which has far fewer semantic possibilities.

Finally, the fourth type of problem mentioned above occurs when heuristics are used to infer a value for an attribute. The pertinent example was given earlier in the case of determining how a student gets to school based on the distance they must traverse--the rule works frequently, but not always. Thus, we must be apprehensive, to a degree, when trying to identify an entity by comparing values that were generated by a heuristic rule. Of course, some rules are stronger in this sense than others: the rules which determine whether a student graduates are always true, by definition, so we would be quite confident in using an inferred value for graduation status to identify a student entity.

If we can represent knowledge about this type of uncertainty we will be able to understand which attribute mismatches we should be concerned about, and which we shouldn't. This, in turn, will allow us to present the possibilities to the user in an informed and efficient fashion. From the above analysis, it is clear that the uncertainty associated with heuristics is only one of a set of similar types of uncertainty associated with a distributed database environment. Thus, it is sensible to treat all types of uncertainty in a consistent fashion. Therefore, we can use a new facet, called a PROBABILITY facet, to represent the uncertainty associated with the timing and semantic problems as well as heuristics. The value of the PROBABILITY facet should be set to a value between 0 and 1 according to the probability that data for the same entity will be recorded the same (absent domain mapping problems) in two different databases given the consideration of the timing and semantic problems. For example, the PROBABILITY facet for age might be set to 0.90 because it changes over time and thus might be different in databases which are updated differently, while the PROBABILITY facet for birthdate might be set to 0.99 since this doesn't change over time and means the same thing to everybody (we choose 0.99 instead of 1.00 only to allow for data problems such as people lying about their age). After the values for the PROBABILITY facet have been set the heuristic rule set is run to infer additional values. The rule set may also modify the values in the PROBABILITY, but should do so in a multiplicative fashion so as not to mask the uncertainty effects of timing and semantics.

Now that we have captured the knowledge about the uncertainty of the attribute values, we can devise a method of ranking the possibilities from those most likely to be the match to those least likely. First, we will check each slot which contains values for attributes which are either in both databases, or in one and for which we can infer a value in the other, based on data in that database. For each slot that we check we will calculate a result: 1, if the slot values match, or (1 - PROBABILITY facet value) if they don't match. The total score for that entity will be the product of each slot result. The entities will then be ranked by score in descending order. All scores will thus range between 0 and 1. If an entity matches on all attibute values it will have a score of 1, and will be presented first to the user. If an entity mismatches on an uncertain attribute like age it will have little effect on the score, whereas if it mismatches on a significant attribute like birthdate it will lower the score greatly. Thus, this procedure effectively incorporates the knowledge that we have elicited about the concreteness of data values.

Now we can present the whole attribute subsetting process. First, a primary-foreign key join is attempted. If unsuccessful, then we try to match unique keys in an IDIIT. If that is unsuccessful, then we begin attribute subsetting. First, the values for the PROBABILITY facets are set for all slots which will participate in the attribute subsetting process. Next, the heuristic rule set is run to infer additional values for attributes which are not contained in one of the databases, as well as to adjust the values of the PROBABILITY facets. Then, all the possibilities are compared and ranked. The ranking is done on the basis of a score which is compiled by multiplying the result of each slot comparison, where the slot result equals one if the values matched, and one minus the PROBABILITY facet value if the slots didn't match. Finally, the ordered list of possibilities can be presented to the user for confirmation of the entity instance identification. If desired, this match can then be placed in an IDIIT for ease of future access. This procedure will be exemplified in the following scenario.

III. A Detailed Scenario

Figure 3 shows a CIS system which links the databases for a Bloodmobile and a University. In this scenario, the Bloodmobile has recently completed a marketing study which showed that college students were their strongest donors, and frequently would return every few months, if contacted. As a result, the Bloodmobile has engineered a partnership with local Universities in order to reach out to their students. In particular, assume that the MIT hospital has made certain of its databases available to the Bloodmobile. The Bloodmobile is interested in ascertaining which of the MIT students have donated blood before. From the MIT database they hope to get the student's current address as well as his health status--they don't want to contact any students who have recently had a blood disease. The problem is that there is no common key for the Bloodmobile to identify students in the MIT database.

BLOODMOBILE

Name	Blood Type	Age	Address	Weight
D.S. Jones	O +	24	Somerville	133
K.C. Tierney	B -	20	Cambridge	124

MIT HOSPITAL

Student ID	Status	Blood Type	Housing
012478430	U	O +	on-campus
261785980	U	A -	off
482947562	G	O +	off
874930193	G	B -	on-campus
364728903	G	B -	off

D.S. Jones

Score	Ranking
(1-.855)(1-.855)	2
(1-.855)(1-.99)	4
1	1
(1-.99)(1-.855)	4
(1-.99)	3

HEURISTICS

IF (age <21) THEN status = "U"
ELSE status = "G"

PROBABILITY = .90

IF (housing = "on-campus") THEN address = "Cambridge"

PROBABILITY = .95

IF (address ≠ "Cambridge") THEN housing "off"

PROBABILITY = .95

Attribute	Probability facet initialized to	After heuristics
Status	.95	.855
Blood Type	.99	.990
Housing	.90	.855

Figure 3.

However, because there are other shared attributes, we can use the attribute subsetting process. In particular, both databases have information on student's blood type. Furthermore, from heuristics, we can infer values for grad/undergraduate status, and, in certain cases, for housing and address. The procedure is as detailed above. First, the values for the attributes that will be compared are set. Blood type is an example of a very strong attribute to use for comparisons: it doesn't change over time, and it is universally understood to mean one thing. Thus, we initially set its PROBABILITY facet value to 0.99. Status is also a fairly strong attribute, but it does change over time and people who are simultaneously in grad and undergrad programs may be registered as either. Thus, we assign a PROBABILITY facet value of 0.90, figuring that 90% of the time different databases will represent a given student entity identically. Finally, housing is a less desirable attribute to make comparisons on because people frequently move, sometimes without updating MIT on their new address. Thus, the PROBABILITY facet for housing is set to 0.90.

Next, the heuristic rule set is run. In this case we are first trying to match the student known to the Bloodmobile as D.S. Jones. By using the heuristics we are able to infer values for status and housing. Note that these heuristics also change the values in the corresponding PROBABILITY facets in the multiplicative fashion described above. Now we are able to compare and rank each of the students in the MIT Hospital database. One instance of the student object is created for the D.S. Jones entity in the Bloodmobile database, and another which will hold each of the entities in the MIT database, one at a time. The first student in the MIT database matches on blood type, but not on status or housing. The score is constructed as noted above and recorded in the table to the right of the MIT Hospital table. The same procedure is followed for each of the other possibilities. After they are all scored, they are given a ranking, also shown in the table to the right of the MIT Hospital table.

In this case, the outcome is easy since there is only one candidate who matched on all the comparison slots. The selected student (D.S. Jones) could then be placed in a student IDIIT table with the corresponding ID (482947562). However, what if that student id and corresponding information didn't exist ? Which would be the most likely candidate to return to the user ? In this case, there is only one which had just one mismatch: however, this was on blood type which is strong evidence of a different entity. Thus, his score was much lower than the others. In fact, one of the possibilities which had *two* mismatches would be returned as our next choice. Even though he mismatched on both status and housing, this was not thought to be as significant since both had a fairly high degree of uncertainty associated with comparisons between different databases.

Thus, this methodology allows us to evaluate different possibilities in a knowledgeable fashion. We are even close to being in a position of being able to answer the user's question of why certain entities seem more likely to match than others. Beyond this, of course, is the possibility of allowing the user to adjust the heuristics as he goes, so that the first few matches would fine-tune the rules for his specific situation. Finally, although we have not addressed how to handle outright data error, this framework could support approaches towards doing that. If, for instance, it was known that certain databases were known to provide "bad" data for certain attributes, then a set of rules could be designed to alter the PROBABILITY facet value based on where the data was retrieved from. This set of rules would then be run after the heuristic rules and would alter the PROBABILITY facet value in the same multiplicative fashion.

Future Directions

Work is currently underway to implement the design that has been laid out in this document. The Local and Global Query Processors described in Chapter 2 have already been built. The translation class objects and attribute subsetting procedures will be next implemented in order to extend the logical connectivity capabilities of the system.

This thesis has focused on the internals of the CIS/TK system. Equally important, however, is the user interface to the system. Future work will be aimed at enhancing the builder interface already designed by Levine[1988]. This will allow system designers to create and view the objects which constitute a Global Schema in a more user-friendly fashion. An SQL-type syntax parser will also be implemented as a query interface for end-users experienced in traditional relational database access. The query syntax presented in this thesis will be retained as the internal query representation.

The design and implementation of CIS/TK will continue to be an iterative process. One area that will be given more thought is how to more fully integrate the rule system into CIS/TK. Currently, rules are used primarily to support heuristics. In the future, rules should be able to more directly manage logical connectivity processes of choosing databases, accessing data, handling credibility issues, etc.. This implies beginning to define a more structured object framework for referring to databases, concepts and rules themselves.

Bibliography

[BAT86] Batini, C., M. Lenzerini, and S.B. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration."

[BEN88] Benjamin, Robert I., et al. "The Realities of Electronic Data Interchange: How Much Competitive Advantage?"

[BRO75] Brooks, Frederick P. The Mythical Man-Month: Essays on Software Engineering. Reading, MA: Addison-Wesley, 1975.

[ELM87] Elmasri, R., J. Larson, and S. Navathe. "Schema Integration Algorithms for Federated Databases and Logical Database Design," Submitted for Publication, 1987.

[LEV87] Levine, Samuel P. "Interfacing Objects and Databases," Master's Thesis, M.I.T. 1987.

[MAD88a] Madnick, Stuart E. and Y. Richard Wang. "Facilitating Connectivity in Composite Information Systems."

[MAD88b] Madnick, Stuart E. and Y. Richard Wang. "A Framework of Composite Information Systems for Strategic Advantage," in Proceedings of the Twenty-first Annual Hawaii International Conference on System Sciences, Vol. III, January 1988, pp. 35-43.

[MAD88c] Madnick, Stuart E. and Y. Richard Wang. "Integrating Disparate Databases for Composite Answers, " in Proceedings of the Twenty-first Annual Hawaii International Conference on System Sciences, Vol. II, January 1988, pp. 583-592.

[MAD88d] Madnick, Stuart E. and Y. Richard Wang. "A Tool Kit for Composite Information Systems: Research Overview, Current Status, & Near-Term Plan," Sloan School of Management Working Paper

[MAS87a] Massachusetts Institute of Technology Sloan School of Management, 1987 Placement Manual.

[MAS87b] Massachusetts Institute of Technology Sloan School of Management, 1987 Placement Report.

[OSB87] Osborn, Charley. "Towards a CIS Model for Strategic Applications."

[TEN88] Tener, Lisa. (Title of report to be determined).

[TRI87] Trice, Andrew, et al. "Placement Office Requirements."

[WEI76] Weick, Karl E. "Educational Organizations as Loosely Coupled Systems."