Towards a Better Understanding of Data Models Through the Multilingual Database System

STEVEN A. DEMURJIAN, MEMBER, IEEE, AND DAVID K. HSIAO, SENIOR MEMBER, IEEE

Abstract—Over the past 20 years, many different data models have been developed, ranging from traditional data models, such as the relational, the hierarchical, and the network models, to newer, semantic data models, such as the functional data model and the entity-relationship model. In spite of the wide range of data models, techniques which can be used to understand data models and data-model semantics by comparing and contrasting models are not available. In order to better understand data models and data-model semantics, a new and unconventional approach to the design of a database system, the multilingual database system (MLDS), has been proposed and implemented. MLDS is a single database system that can execute many transactions written respectively in different data languages and support many databases structured correspondingly in various data models, i.e., DL/I transactions on hierarchical databases, CODASYL-DML transactions on network databases, SQL transactions on relational databases, and Daplex transactions on functional databases. MLDS, by providing an integrated environment for experimenting with data models and data languages, also serves as a test-bed through which we can gain new insight on data models and data-model semantics.

Index Terms—Database system, data-model comparisons, data models, data-model semantics, data-model transformations.

I. INTRODUCTION

DATA models, data languages, and database systems have evolved over a number of years. In the 1960's, IBM introduced the Information Management System (IMS), which supports the hierarchical data model and the hierarchical-model-based data language, Data Language I (DL/I). In the 1970's, Sperry Univac introduced the DMS-1100 database system, offering the network data model and the network-model-based data language, CODASYL Data Manipulation Language (DML). The evolution continued with IBM's introduction of the SQL/Data System in the 1980's which supports the relational model and the relational-model-based data language, Structured English Query Language (SQL) and with CCA's introduction of the Local Data Manager (LDM) database system supporting the Daplex data language based on the functional data model. As in the evolution of software-laden database systems, the hardware-assisted database systems followed the same pattern. Thus, the experimental CASSM [34] database machine of the seventies supported the hierarchical data model and data language. The prototyped XDMS [10] database backend also of the seventies supported the network data model and CODASYL-DML. More recently, the Britton-Lee IDM/500 and the Teradata DBC/1012 support the relational data model and relational-model-based data languages similar to SQL.

What we have seen in this evolution is the development and proliferation of a variety of database systems based on many different data models. The data models range from well-established, traditional data models such as the relational model, the hierarchical model and the network model, to newer, semantic data models, such as the functional model and the entity-relationship model. Despite all of the work that has been conducted on data models, there has been very little work done on data-model comparisons and analyses.

From a data-model perspective, we would like to have concrete, qualitative and quantitative techniques to compare and contrast the features of the different data models, in order to arrive at an understanding of the data-model semantics. However, we note that there has been very little work done on data-model comparisons and analyses. Brodie [9] has presented a taxonomy of the three classical data models (i.e., relational, hierarchical, and network). However, the taxonomy is very broadly based and simply compares the features and characteristics of the three classical data models. Instead, much of the work in this area has focused on formalization techniques for specifying data models. The work of Bjorner [8] has been in the programming language specification technique of denotational semantics to abstractly specify data models. The work of Abiteboul and Hull [1] has presented the formalization of a new semantic data model, called IFO, that is capable of capturing the relationships of other, more classical data models. While we feel that their work is important, we also believe that there is a research gap present in the work on data models, namely, techniques which can be used to analytically compare and contrast existing data models.

Therefore, in order to better understand data models and data-model semantics, a new and unconventional approach to the design of a database system, the multilin-
that can execute many transactions written respectively in
different data languages and support many databases
gual database system
through which we can gain new insight on data models
by providing an integrated environment for experimenting
tional databases, where the system appears to the user like
transactions on hierarchical databases, CODASYL-DML
example, the multilingual database system can run DL/I
been developed using MLDS to compare and contrast data
models and data-model semantics. More specifically, MLDS
serves as the means through which we can develop qual-
itive and qualitative techniques that can be used to
compare and contrast data models, and thereby ultimately
lead to a better understanding of models.

The remainder of this paper is organized as follows. In
Section II we describe the multilingual database system,
 focusing on its motivation and structure. In Section III,
we present an overview of how MLDS can be used as a
test-bed to attain a better understanding of data models
and data-model semantics. In particular, we briefly pre-
sent the quantitative and qualitative techniques that have
been developed using MLDS to compare and contrast data
models. Finally, in Section IV, we conclude this paper,
indicate our other related work on data-language compar-
ison and analysis, and speculate on our future work in
MLDS.

II. THE MULTILINGUAL DATABASE SYSTEM

While MLDS supports many data models and data lan-
guages, the underlying database system that is part of
MLDS supports only a single data model and data lan-
guage. Referred to as the kernel data model (KDM) and
kernel data language (KDL), the single data model and
language comprise the physical database structure and
representation for MLDS. In searching for a kernel data
model and kernel data language with a high-level struc-
ture, which will support different data models and data
languages, we are examining the transformations of var-
ious data models into the kernel data model and the trans-
lations of various data languages to the kernel data lan-
guage. The mapping process from a given data model to
KDM is called data-model transformation. The mapping
process from a given data language to KDL is called data-
language translation. To design a multilingual database
system, the data-model transformations and data-lan-
guage translations must be specified. By specifying the
various data-model transformations (e.g., from the rela-
tional model to the KDM, from the hierarchical model to
KDM, from the network model to the KDM, and from the
functional model to the KDM), we provide a single da-
tabase system that is capable of supporting many models
on a kernel data model. Similarly, by providing the vari-
ous data-language translations (e.g., from SQL to KDL,
from DL/I to KDL, from CODASYL-DML to KDL, and
from Daplex to KDL), we also provide that database sys-

From the practical merits
of MLDS, we are able to demonstrate the

carbon and useful features of such a system. Second, by
identifying the new functionalities inherent in MLDS, we
are able to provide the incentives for the user to move
from a conventional database system to MLDS. Third,
and most important, by verifying the theoretical issues
required to support multiple data models and data lan-
guages in MLDS, we may gain a better understanding into
the structures of and relationships among different data
models and data languages.

There are three main practical merits associated with
the multilingual database system. First, there is the abil-
ity to reuse database transactions. Since the multilingual
database system supports many different data models and
their data languages, database transactions developed on
another database system can easily be transported to the
multilingual database system. Thus, old applications can
still be supported in the new environment. Second, there
is the economy and effectiveness of the hardware up-
grade, software maintenance and manpower support.
Since there is only a single multilingual database system
to maintain, rather than three or four separate, monolin-
gual database systems, the costs associated with purchas-
ing hardware and software, maintaining the database sys-
tem software, and providing in-house software support are
significantly reduced. Finally, there is the uniform exe-
cution environment of the database system. Since there is
a single database system, the user is only required to learn
a single database command language to access the data-
base system. This is in contrast to the three or four dif-

A. Issues and Merits of MLDS

While the need for providing a wide-range of applica-
tions on a single database system is important, there are
a number of other advantages and benefits to developing
a multilingual database system. First, by studying the
practical merits of MLDS, we are able to demonstrate the
concrete and useful features of such a system. Second, by
identifying the new functionalities inherent in MLDS, we
are able to provide the incentives for the user to move
from a conventional database system to MLDS. Third,
and most important, by verifying the theoretical issues
required to support multiple data models and data lan-
guages in MLDS, we may gain a better understanding into
the structures of and relationships among different data
models and data languages.

The remainder of this section, we first review some of
the advantages and benefits for developing MLDS.
Next, a brief presentation of the system structure of
MLDS is given. Third, we present related work in sup-
porting other data models and data languages on a single
database system. Fourth, we examine and justify our
choices for the kernel data model and kernel data lan-
guage. Finally, we provide an example of the data-model
transformation and data-language-translation processes,

A. Issues and Merits of MLDS

While the need for providing a wide-range of applica-
tions on a single database system is important, there are
a number of other advantages and benefits to developing
a multilingual database system. First, by studying the
practical merits of MLDS, we are able to demonstrate the
concrete and useful features of such a system. Second, by
identifying the new functionalities inherent in MLDS, we
are able to provide the incentives for the user to move
from a conventional database system to MLDS. Third,
and most important, by verifying the theoretical issues
required to support multiple data models and data lan-
guages in MLDS, we may gain a better understanding into
the structures of and relationships among different data
models and data languages.

There are three main practical merits associated with
the multilingual database system. First, there is the abil-
ity to reuse database transactions. Since the multilingual
database system supports many different data models and
their data languages, database transactions developed on
another database system can easily be transported to the
multilingual database system. Thus, old applications can
still be supported in the new environment. Second, there
is the economy and effectiveness of the hardware up-
grade, software maintenance and manpower support.
Since there is only a single multilingual database system
to maintain, rather than three or four separate, monolin-
gual database systems, the costs associated with purchas-
ing hardware and software, maintaining the database sys-
tem software, and providing in-house software support are
significantly reduced. Finally, there is the uniform exe-
cution environment of the database system. Since there is
a single database system, the user is only required to learn
a single database command language to access the data-
base system. This is in contrast to the three or four dif-

A. Issues and Merits of MLDS

While the need for providing a wide-range of applica-
tions on a single database system is important, there are
a number of other advantages and benefits to developing
a multilingual database system. First, by studying the
practical merits of MLDS, we are able to demonstrate the
concrete and useful features of such a system. Second, by
identifying the new functionalities inherent in MLDS, we
are able to provide the incentives for the user to move
from a conventional database system to MLDS. Third,
and most important, by verifying the theoretical issues
required to support multiple data models and data lan-
guages in MLDS, we may gain a better understanding into
the structures of and relationships among different data
models and data languages.

There are three main practical merits associated with
the multilingual database system. First, there is the abil-
ity to reuse database transactions. Since the multilingual
database system supports many different data models and
their data languages, database transactions developed on
another database system can easily be transported to the
multilingual database system. Thus, old applications can
still be supported in the new environment. Second, there
is the economy and effectiveness of the hardware up-
grade, software maintenance and manpower support.
Since there is only a single multilingual database system
to maintain, rather than three or four separate, monolin-
gual database systems, the costs associated with purchas-
ing hardware and software, maintaining the database sys-
tem software, and providing in-house software support are
significantly reduced. Finally, there is the uniform exe-
cution environment of the database system. Since there is
a single database system, the user is only required to learn
a single database command language to access the data-
base system. This is in contrast to the three or four dif-

A. Issues and Merits of MLDS

While the need for providing a wide-range of applica-
tions on a single database system is important, there are
a number of other advantages and benefits to developing
a multilingual database system. First, by studying the
practical merits of MLDS, we are able to demonstrate the
concrete and useful features of such a system. Second, by
identifying the new functionalities inherent in MLDS, we
are able to provide the incentives for the user to move
from a conventional database system to MLDS. Third,
and most important, by verifying the theoretical issues
required to support multiple data models and data lan-
guages in MLDS, we may gain a better understanding into
the structures of and relationships among different data
models and data languages.

There are three main practical merits associated with
the multilingual database system. First, there is the abil-
ity to reuse database transactions. Since the multilingual
database system supports many different data models and
their data languages, database transactions developed on
another database system can easily be transported to the
multilingual database system. Thus, old applications can
still be supported in the new environment. Second, there
is the economy and effectiveness of the hardware up-
grade, software maintenance and manpower support.
Since there is only a single multilingual database system
to maintain, rather than three or four separate, monolin-
gual database systems, the costs associated with purchas-
ing hardware and software, maintaining the database sys-
tem software, and providing in-house software support are
significantly reduced. Finally, there is the uniform exe-
cution environment of the database system. Since there is
a single database system, the user is only required to learn
a single database command language to access the data-
base system. This is in contrast to the three or four dif-
There are two major new functionalities offered by the multilingual database system. First, since many different models and their languages are supported, the multilingual database system provides an environment that is conducive to the experimentation on and investigation of the many models and languages. From this perspective, the multilingual database system is an excellent tool for teaching students about the different data models, data languages, and database applications. Second, the multilingual database system can also be used as a rapid prototyping system. In this case, we can augment the functionality of the multilingual database system by adding another data model/data language combination to the system more easily than we can construct a new monolingual database system. This allows us an improved capability to explore new data models, data languages, and database applications.

Finally and most importantly, the theoretical issues of the multilingual database system are in its ability to serve as a test-bed for the investigation of and experimentation on many different data models and their data languages. In this case, we are interested in learning how data models are related by an examination of data-model semantics. Thus, we have developed quantitative and qualitative techniques for examining the relationships between data models by using the data-model-transformation processes. These techniques are discussed in Section III of this paper. We are also interested in determining the relationships that exist between data languages. So, we have developed a quantitative technique for examining the relationships between data languages by using the data-language-translation processes [15]. In both cases, by providing an integrated environment where many models and languages are supported on a single database system, we have developed a vehicle for comparing data models and data languages.

B. The Organization of MLDS

The system structure of a multilingual database system is shown in Fig. 1. Users issue transactions through the language interface layer (LIL) using a user-chosen data model (UDM) and written in a corresponding model-based data language (UDL). LIL then routes the user transactions to the kernel mapping subsystem (KMS). KMS has two tasks. First, if the user specifies that a new database is to be created, KMS transforms the UDM-database definition to an equivalent kernel-data-model (KDM)-database definition. The KDM-database definition is then sent to the kernel controller subsystem (KCS). KCS sends the KDM-database definition to the kernel database system (KDS). Upon completion, KDS notifies KCS, which in turn, notifies the user that the database definition has been processed and that the loading of the database may commence.

The second task of KMS is to handle UDL transactions. In this situation, KMS translates the UDL transaction to an equivalent kernel-data-language (KDL) transaction. KMS then sends the KDL transaction to KCS, which in turn, sends the KDL transaction to KDS for execution. Upon completion, KDS sends the results in the KDM form back to KCS. KCS forwards these results to the kernel formatting subsystem (KFS) for transforming them from the KDM form to the UDM form. After the data is transformed, KFS returns the results, i.e., the response set, to the user via LIL.

There is one final note of importance on the general system structure. Four of the five components of the multilingual database system, namely, LIL, KMS, KCS, and KFS, are referred to as a language interface. A new language interface is required for each chosen data language. For example, there is a set of LIL, KMS, KCS, and KFS for the relational/SQL language interface, a separate set of these four components for the hierarchical/DL/I language interface, a third set of components for the network/CODASYL-DML language interface, and a fourth set for the functional/Daplex language interface. KDS, on the other hand, is a single and major component that is accessed and shared by all of the various language interfaces as also depicted in Fig. 1.

C. Other Approaches to Data-Model Transformation and Data-Language Translation

In the areas of data-model transformation and data-language translation, there are other efforts. In one effort, the goal is to examine the capability of an existing database system in supporting another data model and language on the existing system. The work of Katz [23] and Demo [13], [14] supports the network data model and CODA-
SYL-DML data language on a relational system. Furthermore, the support of the relational data model and data language on a network system and the support of the network data model and data language on a relational database system have been examined by Larson [26]. Each of these examinations is essentially restricted to the mapping from one data model and data language to another data model and data language. We refer to their approaches as the single-ML-to-single-ML mapping, where ML stands for model and language.

The other effort in data-model transformation and data-language translation focuses on communicating with a heterogeneous collection of separate database systems via a local-area network. In this effort, a global data model and global data language are defined. By using a global data model and global data language, the user is able to obtain uniform access to a number of database systems based on different data models and data languages [20]. The CCA Multibase System [32], UCLA DBMS [11], SDC Mermaid [35], and NBS XDMS [24] are examples of database systems each of which maps a single (global) data model and single (global) data language to a collection of data models and data languages, e.g., the single-ML-to-multiple-ML mapping.

It is interesting to note that MLDS described herein maps respectively many different data models and data languages to a single (kernel) data model and a single (kernel) data language, which is the multiple-ML-to-single-ML mapping, the third broad class of mappings. The single-ML-to-multiple-ML mapping is the reverse of the multiple-ML-to-single-ML mapping and the single-ML-to-single-ML mapping is a special case of the more general case of the multiple-ML-to-single-ML mapping. In other words, the CCA work, for example, is the reverse of our work on MLDS, and Katz’s work and Demo’s work are both special cases of our more general work on MLDS. There have been other efforts to support multiple-ML-to-single-ML mapping. The work on the Damstadt database kernel system [29] is developing a system with interfaces for geoscience applications, office-document filing, and standard relational applications. Also, we note that in Demo [13], there is a mention that their research group plans on examining the support of the hierarchical data model and DL/I on a relational system. Such an investigation would move their system from the single-ML-to-single-ML mapping class to the multiple-ML-to-single-ML mapping class.

D. Choosing a Kernel Data Model and Language

As a prerequisite for examining our experience with MLDS, we first explore the two goals of the mapping process, i.e., data-model transformation and data-language translation. In data-model transformation, we must be sure that the data semantics are preserved. When converting a database (modeled in, for example, one of the four aforementioned models) to a kernel database, we must ensure that an equivalent and complete database can be created. In other words, the source (user) database and the transformed target (kernel) database have the same semantics. In data-language translation, we must be sure that the translated transaction operations are equivalent. Thus, when translating a source (user) transaction (written in, for example, one of the four aforementioned data languages) to a target (kernel) transaction written in the kernel data language, we must ensure that the access of the stored database by the target transaction results in the correct action on the database as required by the source transaction. Consequently, semantic preservation of the database and operational equivalence of the transactions are the goals of the data-model transformation and the data-language translation, respectively.

To us, the key decision in the development of a multilingual database system is therefore the choice of a kernel data model and kernel-model-based data language, so that semantic preservation and operational equivalence can be facilitated with ease. In our effort, we experiment with the attribute-based data model proposed by Hsiao [22], extended by Wong [38], and studied by Rothnie [33], as the kernel data model. In the attribute-based data model, databases are defined by specifying a collection of attributes that are in the database. The database records in the attribute-based data model consist of sets of attribute-value pairs. An attribute-value pair or keyword is a member of the Cartesian product of the attribute name and the value domain of the attribute. For example, <Population, 25000> is an attribute-value pair having 25,000 as the value for the population attribute, while <Population, 12000> is an attribute-value pair having 12,000 as the value for the population attribute. The number of different attribute-value pairs is limited by the cardinality of the value domain for the given attribute. Database records have the property that for a given set of attribute-value pairs, no two pairs in the set have the same attribute name. Certain attributes are directory attributes and their keywords, directory keywords. Database operations to access and modify the database are performed on the database records via the predicates of directory and nondirectory keywords.

The attribute-based data language (ABDL) defined in Banerjee [3] and extended by Tung [36] is chosen as the kernel data language. The attribute-based data language supports the five primary database operations, INSERT, DELETE, UPDATE, RETRIEVE, and RETRIEVE-COMMON. A request in the ABDL is a primary operation with a qualification. A qualification is used to specify the part of the database that is to be operated on. Two or more requests may be grouped together to form a transaction. The RETRIEVE-COMMON operation is used to provide the merge of two files with common attribute values, of which a relational equijoin is a special case. Together, ABDL provides five seemingly simple database operations, which are nevertheless capable of supporting complex and comprehensive transactions.

The main question is whether or not the attribute-based data model and data language are capable of supporting the required data-model transformations and data-lang-
guage translations. Is it easy to transform a relational, hierarchical, network or functional database to an attribute-based database with the data semantics intact? Can SQL, DL/I, CODASYL-DML, and Duplex operations be translated easily to ABDL operations with the transaction operations being equivalent?

The series of papers [4], [5], [6] have shown how the relational, hierarchical, and network data can be transformed to attribute-based data and also presented some preliminary work on the corresponding data-language translations. More recently, the complete sets of algorithms for the data-language translation from SQL to ABDL [28], [31], from DL/I to ABDL [37], from CODASYL-DML to ABDL [39], and from Duplex to ABDL [21], have been specified. Software development efforts for language interfaces (i.e., one set of LIL, KMS, KFS, and KCS for the relational interface [7], another set for the hierarchical interface [25], a third set for the network interface [19], and a fourth set for the functional interface [2], [27]), have been completed.

Another, equally important reason for choosing the attribute-based data model as the kernel data model and ABDL as the kernel data language lies in the availability of a research database system in the Laboratory for Database Systems Research. This database system, the multibackend database system (MBDS), uses, respectively, the attribute-based data model and ABDL as the native data model and data language of the system. Thus, MBDS can serve as an ideal test bed for KDS. The interested reader may refer to Demurjian and Hsiao [16], [17] for an overview of the structure and operation of MBDS.

E. An Example of Transformation and Translation

In this section we provide an example of both a data-model transformation (i.e., from hierarchical to attribute-based) and a data-language translation (i.e., from DL/I to ABDL). For the data-model transformation, we stress how we obtain the semantic preservation of UDM in KDM. For the data-language translation, we provide some insight into how KCS insures the semantic preservation of UDM in KDM, achieves the operational equivalence of UDL in KDL, and mimics the UDM-UDL execution environment. Due to obvious space limitations, we do not include herein our other work on data-model transformations (i.e., from relational to attribute-based, from network to attribute-based, and from functional to attribute-based). Nor do we include herein our other work on data-language translation (i.e., from SQL to ABDL, from CODASYL-DML to ABDL, and from Duplex to ABDL).

The reader is referred to Demurjian [15] for a complete and formal presentation of data-model transformation and data-language translation techniques.

1) Hierarchical Data to Attribute-Based Data: Hierarchical data is organized into occurrences of segment types or briefly, segments. A database is a collection of segments, structured in a hierarchical or tree-like fashion. These segments have the property that no two occurrences are identical. The fields of a segment type are distinct. Additionally, an occurrence in one segment type may correspond to one or more occurrences of another segment type. Such a correspondence is often characterized as a parent-child relationship between the two segment types, i.e., a one-to-many relationship. A definition of the Course-Prereq-Offering hierarchical database is given in Fig. 2.

In the hierarchical version of the database, courses are uniquely identified by course numbers and have titles and descriptions, i.e., the Course (root) segments. The unique identifiers are used both as a key and for sequencing. Each course has one or more prerequisite courses (i.e., a one-to-many relationship), as indicated by the arrow between the Course and Prereq segments in Fig. 2. Similarly, each course has one or more dates on which the course is offered (i.e., also a one-to-many relationship), as indicated by the arrow from the Course segment to the Offering segment.

The mapping of the hierarchical data to the attribute-based data is straightforward. The concepts of a database, a segment type, and an occurrence in the hierarchical data correspond to a database, a file, and a record in the attribute-based data. In this way, the hierarchical database shown in Fig. 2 can be mapped to the attribute-based database given in Fig. 3.

In Fig. 3, each segment type in the database has been transformed into a collection of attribute-based keywords. Unlike the hierarchical specification in Fig. 2, where the values of the fields need not appear, the attribute-based specification in Fig. 3 requires a place holder in each keyword for the attribute value. This place holder is used to present the corresponding attribute-value type of the attribute value. For the hierarchical transformation, the possible attribute-value types are string, integer, floating-point number, io-value, and r-value.

The hierarchical to attribute-based data-model transformation occurs in a segment-by-segment fashion. In Fig. 3, we see that each field of a segment occurrence has been transformed into a keyword of a record. In order to differentiate between segments, the segment name has been included as an attribute value in the File keyword. Thus, we encode the segment (type) name as a File keyword. We have also included a keyword for each field in a segment, i.e., the Course#, Title, and Descrip keywords in the Course file. However, we notice that while the Title and Desc keywords have typical value types (i.e., string) the value type for the sequence field of the segment (i.e., Course#) is of io-value. The "io" in io-value stands for identification/ordering. As mentioned earlier, in a hierarchical segment, the sequence field is used to uniquely identify every segment occurrence in the segment type. Hence, the "i" denotes this situation. Also, there is an ordering that is enforced for all records in a segment type, namely, the ordering of the child records in a parent-child relationship. In this case, the "o" represents the ordering aspect of the sequence field.

Besides the keywords for the segment name and the fields of the segment type, there are also some other key-
words present in the transformation of the Prereq and Offering segment types. First, we see that the sequence field for the Course segment type has been cascaded into the attribute-based record types for Prereq and Offering. The keyword added for each of these two attribute-based record types, namely, Prereq, r-value>, is introduced to capture the one-to-many relationship between the Course (root) segment and its two children. The ‘r’ in r-value represents the word referential, since the actual value refers to a record in the Course root segment. By having the Course# keyword in every occurrence in the Offering (Prereq) segment, we establish the parent-child correspondence between an occurrence in the Course segment and its children in the Offering (Prereq) segment. By cascading the sequence fields of the hierarchical segments into the attribute-based records, we capture the unique identification of each segment occurrence which is required by the hierarchical model.

Finally, we note that in our conceptualization and implementation of the hierarchical/DL/I language interface, we have assumed that, for the model, each hierarchical segment type has exactly one sequence field. If a segment type had two or more sequence fields, we could always transform that type to an equivalent type with exactly one sequence field. The new single sequence field has an artificial domain which is defined over the domains of the multiple sequence fields in the original segment type. Likewise, if a segment type does not have a sequence field, we can simply introduce an artificial sequence field to serve our purposes. At this time, we also note that we have made similar assumptions concerning the relational model (i.e., only a single key in a relation).

2) DL/I Requests to ABDL Requests: We now present an overview of the data-language translation from DL/I GU, GNP, and GN statements to ABDL RETRIEVE requests. Our major emphasis in this section is examining the execution sequence and responsibilities of KCS to insure operational equivalence. The complete data-language translation from DL/I to ABDL is documented by Weishar [37]. The implementation of the DL/I language interface, is reported by Benson and Wentz [7].

The general form of a DL/I GU or GN differs only in the operation type, and is shown below. The dml-operator can be either GU, GN or any of the other DL/I operations.

dml-operator seg-srch-arg [seg-srch-arg-list]

To access a segment occurrence, we specify the segment in a predicate known as the segment-search argument (seg-srch-arg), which includes the segment name and an optional boolean expression of predicates. The segment-search-argument list (seg-srch-arg-list) is an optional list of one or more segment-search arguments.

When a DL/I GU, GNP, or GN is translated, one or more ABDL RETRIEVE requests may result. For example, consider the DL/I GU and GNP request shown below.

GU Course (Title = Advanced Database Systems)

LOOP GNP Prereq GOTO LOOP

This DL/I GU and GNP retrieves all prerequisites for the advanced-database-systems course. In translating this statement, two ABDL RETRIEVE requests, R1 and R2, are generated:

R1 [RETRIEVE ((File = Course) and (Title = Advanced Database Systems)) (Course#) BY Course#]

R2 [RETRIEVE ((File = Prereq) and (Course# = ***)) (Pcourse#, Title) By Pcourse#]

The sorting in R1, via the BY clause, is used to simulate the hierarchical structure of the data in an IMS system. This in turn helps insure the operational equivalence of the ABDL requests. In the general case, an n-level DL/I GU, GNP or GN translates to n ABDL RETRIEVE requests.

The emphasis here is on the execution sequence of requests R1 and R2 by KCS. KCS begins by executing R1. When R1 is sent to KDS for execution, a course number corresponding to the advanced-database-systems course is returned in the result buffer as the currency indicator. Say the returned course number is CS4322. KCS takes this course number and reforms the request R2 by inserting CS4322 for the place-holder marked by asterisks. The reformed request R2 is then sent to KDS for processing. Suppose that the result of this request is

Pcourse# Title
CS3450 Systems Programming
CS4300 Intro. Database Systems

Since a GNP with a loop is utilized, KFS forwards all of the results in this buffer to the user. These results correspond to all of the prerequisite courses for the advanced-database-systems course.
The example presented in this section is meant to acquaint the reader with some of the techniques and basic terminology utilized in data-model transformation and data-language translation. In the next section, as we present our work on comparison techniques to better understand data models, we frequently refer back to this example, as a way to demonstrate and explain how the different comparison techniques are related to actual examples.

III. UNDERSTANDING DATA MODELS

We have developed two techniques for comparing data models using their data-model transformations, a quantitative metric and a qualitative method. The goal of these techniques is to allow us to better understand the data models by understanding the different relationships that exist among them. In the next two sections, we discuss each of these techniques in turn. Finally, we note that a complete discussion of our work in using MLDS for understanding data models and data languages can be found in Demurjian [15].

A. A Quantitative Metric for Comparing Data Models

When examining a data-model transformation, we notice that the focal issue is transforming the user data-model constructs to the kernel data-model constructs, in particular, to attribute-based keywords. The attribute-based keywords that are involved in a data-model transformation, can be broadly classified into two categories, basic transformational keywords and acquired transformational keywords. Basic transformational keywords are those keywords which are specified in the transformation as a result of the basic components of the data-grouping relationships of the user data model, for example, keywords for the field names of a hierarchical segment. In Fig. 3, basic transformational keywords would be <Course#, io-value>, <Title, string>, or <Date, io-value>. In our other data-model transformations, the basic transformational keywords would be the attributes of a relational table, the data items of a network record, and the functions of a functional entity type or subtype.

Acquired transformational keywords are extra or special keywords that are required to accomplish a particular transformation and to capture certain data semantics. In Fig. 3, acquired transformational keywords would be the keywords added to represent the segment type (e.g., File, Course), and the keywords which represent the cascaded sequence fields needed to capture the parent-child relationships (i.e., the <Course#, r-value> keyword in the Prereq and Offering files). In our other data-model transformations, the acquired transformational keywords are for the relation (table) names in a relational database, for the record-type names and other special attributes used to represent the set occurrences in network databases, and for the entity-type or subtype names and for representing the generalization hierarchies in functional databases.

We can use these two keyword categories to develop a quantitative metric to measure the complexity of transforming a given UDM database to an attribute-based database. By measuring the complexity of the data-model transformation processes, we can compare the different data models on the basis of their transformations to a common data model. First, we know that the number of basic transformational keywords, while not a constant, is a relatively stable, calculable value. This is because the values for the number of basic transformational keywords in the different transformations correspond to the average number of attributes in a relation times the number of relations, the average number of fields in a segment type times the number of segment types, the average number of data items in a record type times the number of record types, and the average number of functions in entity types or subtypes times the total number of entity types and subtypes. Second, we know that the number of acquired transformational keywords is not a stable value, but is dependent upon the characteristics of the user database (i.e., the characteristics of the relational, hierarchical, network, or functional model of the database in consideration). It is this dependency, which we wish to examine in order to measure the complexity of a data-model transformation. The underlying premise is that we must live with the stable overhead of the number of basic transformational keywords, but the overhead of the number of acquired transformational keywords is very dependent on the characteristics of the built-in relationships of a particular database to be transformed.

To compare the complexities of the four data-model transformations, we introduce the relative percentage of acquired transformational keywords metric. For a given data-model transformation, this metric examines the ratio of the number of acquired transformational keywords and the total number of transformational keywords (both acquired and basic). The formula for the comparative metric is given below:

$$\text{Comparative metric} = \left( \frac{\text{Number of Acquired Transformational Keywords}}{\text{Total Number of Transformational Keywords}} \right) \times 100 \text{ percent.}$$

The comparative metric is used to compare the transformation process itself (i.e., how much work is required to do a particular transformation). For example, in Fig. 3, this ratio would be $5/13 = 38.5$ percent. The premise is that the major effort in a data-model transformation is in representing the data relationships supported by a particular user data model, e.g., the segment-type relationship and the parent-child relationship. This effort is represented by the presence of a high percentage of acquired transformational keywords.

To demonstrate how the comparative metric may be used, we calculate the relative percentage of acquired transformational keywords for four sample data-model transformations [15]. That is, from a given database description, we constructed equivalent databases in rela-
ational, hierarchical, network, and functional form. Once this construction was complete, we performed the data-model transformations and then calculated the ratio (i.e., percentage of acquired transformational keywords), for each of the four different data-model transformations. The calculation is summarized in Table I. What do the percentages in Table I represent? Intuitively, each percentage is a measure of the “extra” or “special” data constructs (semantics) that are in the particular user data model (i.e., relational, hierarchical, network or functional data models) that must be captured by the kernel data model (i.e., the attribute-based data model). Thus, we have the relational model having 25 percent extra or special data constructs (semantics), the hierarchical model having 44 percent extra or special data constructs (semantics), the network model having 56.3 percent extra or special data constructs (semantics) and the functional model having 32 percent extra or special data constructs (semantics).

Overall, as we progress through the data models (i.e., from relational to hierarchical to network to functional) the number of data relationships each model can support increases, so, we expect to find that the comparative metric will show that the complexity of the transformation processes correspondingly increases, i.e., there are more data semantics that require additional extra or special keywords. As expected, we see that the comparative metric for the hierarchical transformation is greater than the metric for the relational transformation, thereby implying that the hierarchical transformation is more complex and, therefore, richer in data semantics. This is expected, since the hierarchical model can support the parent–child and data-sequencing relationships, two data relationships that are not supported by the relational model. Similarly, we find the network transformation more complex than the hierarchical or relational transformations. However, we find that the functional transformation is less complex than the hierarchical or network transformations. Why does this occur?

The greatest portion of acquired transformational keywords in the hierarchical and network transformations are required to represent the parent–child and network–set relationships. For the hierarchical model, to correctly determine the parent of a given segment occurrence, the sequence fields are cascaded down through the hierarchy. For the network model, the network–set relationships between a single owner record and all of its member records must be captured by assigning database keys to records, by cataloging the owner database key using special member keywords, and by ordering all of its member records using special position keywords. However, in the functional model, all of the data relationships are unordered, so that there is no need to enforce ordering. Also, in the functional data model, the data relationships between entity types and subtypes are part of the type or subtype, i.e., they are defined as entity-valued functions in the functional data model, and, as such, are basic transformational keywords. However, in both the hierarchical and the network models, the parent–child and network–set relationships are external to the segment-type and record-type relationships and require acquired transformational keywords, rather than the basic transformational keywords of the functional model. So, in fact, from a purely algorithmic perspective, we would expect that the complexity of the functional transformation as calculated by the comparative metric would be less complex than either the hierarchical or network transformations.

What we do notice is that the comparative metrics for the relational and functional transformations are very close, as are the comparative metrics for the hierarchical and network transformations. In the relational and functional cases, there is no sequencing allowed, so the only difference evident in the metrics is traced to the data relationships (e.g., one-to-many, many-to-many, and generalization) supported in the functional model that are not in the relational model. In the hierarchical and network cases, there is sequencing of the data allowed. Once again, the difference between the two metrics is traced to the data relationships (e.g., many-to-many) supported in the network model that are not in the hierarchical model. However, the difference in the metrics between the relational and functional cases and the hierarchical and network cases is due to the presence of sequencing in the hierarchical and network cases. Therefore, we have that both the data relationships and the sequencing aspects of a data model determine its comparative metric of acquired transformational keywords.

Overall, the comparative metric of complexity can be used to estimate and gauge the complexity of actual database transformations (or data-model semantics) by examining the relative percentage of acquired transformational keywords. The end result of the comparison of the data-model-transformation processes provides us with a total order of data models, with respect to the ratio that represents the complexity of their transformations. The total ordering is:

\[
\text{relational} < \text{functional} < \text{hierarchical} < \text{network}.
\]

Although the ordering is obtained, the quantitative metric that arrived at this ordering may restrict our ability to provide a semantic insight into the relationships between data models. That is, while there may be a relationship between our metric and the semantic content of the data models, we have merely presented some numeric evidence to support this semantic ordering. But, it would be

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Comparative Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>25 percent</td>
</tr>
<tr>
<td>Hierarchical</td>
<td>44 percent</td>
</tr>
<tr>
<td>Network</td>
<td>56.3 percent</td>
</tr>
<tr>
<td>Functional</td>
<td>32 percent</td>
</tr>
</tbody>
</table>
very interesting if we could develop a nonnumerical, analytical approach that could be used to verify or refute the total ordering that is presented above. If we can verify the ordering, we are able to convince ourselves that the metric does indeed provide a measure of the semantic ordering of the data models. If we can refute the ordering, we are still left with a metric that indicates the relative complexity of the different data-model-transformation processes. In the next section, we present a qualitative method that can be used to analyze and compare the data-model transformations, in order to provide a semantic (therefore, complementary) ordering of the data models.

B. A Qualitative Method for Comparing Data Models

In the previous section, we introduced two broad categories of keywords, basic transformational keywords and acquired transformational keywords. In examining these two categories of keywords for each of the four different data-model transformations, we found that there were different types of basic and acquired transformational keywords. For example, in the data-model transformation presented in Fig. 3, basic transformational keywords are used to represent both the sequence field (e.g., <Course#, io-value>) and the nonsequence fields (e.g., <Title, string>) of a hierarchical segment type. However, we notice that, by the way that their values are used, these are two different types of basic transformational keywords, i.e., the Course# values are used for both identification and ordering, while the Title values are only used to store course titles. So, the first step in developing a qualitative method for comparing data models via their transformations, is to completely categorize the different types of basic and acquired transformational keywords.

In our analysis of the four different data-model transformations [15], we have found that these two categories of keywords can be further separated into six keyword categories: type-name-inclusion keywords, elementary keywords, identification keywords, ordering keywords, referential keywords and identification-ordering keywords. Together, these six categories capture all of the possible keywords that may appear in any of our four data-model transformations. Type-name-inclusion keywords are acquired transformational keywords that are used to encode the name of the data-grouping relationship of the user data model in the attribute-based data model. Type-name-inclusion keywords are present in all four of the data-model transformations. For example, in Fig. 3, type-name inclusion keywords are used to encode the segment-type name as an attribute-value in the File keywords, e.g., <File, Course> or <File, Prereq>. Elementary keywords are basic transformational keywords that are used to classify the atomic components of the data-grouping relationships of the user data model in the attribute-based data model. Elementary keywords are present in all four of the data-model transformations. In Fig. 3, the keywords <Title, string> and <Location, string> are examples of elementary keywords.

Identification keywords, which may be either basic or acquired transformational keywords, are used to define a keyword that can be used to uniquely identify the individual members of a data-grouping relationship of the user data model in the attribute-based data model. Identification keywords are present in all but the hierarchical transformation. For example, in the relational data-model transformation, an identification keyword is used for capturing an attribute which serves as the key of a relation for uniquely identifying the tuples. Ordering keywords are acquired transformational keywords that are used to capture the ordering between the data-grouping relationships of the user data model within the attribute-based data model. Ordering keywords are present in only the network transformation. For example, in the network transformation, an ordering keyword is used to represent the ordering of the set members in a set occurrence.

Referential keywords, which may be either basic or acquired transformational keywords, are keywords which are used to reference a data-grouping relationship, rather than an individual component of a data-grouping relationship. Referential keywords are present in all but the relational transformation. In Fig. 3, the keyword for the cascaded sequence field, <Course#, r-value>, in the Prereq and Offering files is an example of a referential keyword. Finally, identification-ordering keywords, which may be either basic or acquired transformational keywords, are a combination of the identification keywords and the ordering keywords. These keywords have a dual purpose, first, to uniquely identify each record in a data-grouping relationship, and, second, to order the data in an data-grouping relationship. Identification-ordering keywords are present in the hierarchical and network transformations. In Fig. 3, the keyword for the sequence field (e.g., <Course#, io-value> or <Date, io-value>) is an example of an identification-ordering keyword.

The second step in our development of a qualitative method to compare data models via data-model transformations is to specify a technique to formalize the data-model-transformation process. Using the six categories of keywords that have been defined, we can group these different categories of keywords together to form what are termed transformation classes. A transformation class consists of the specification of a set of keyword categories. By specifying the set of transformation classes for a particular data-model transformation, we are able to formally and abstractly denote the characteristics of each transformation, independent of any actual database description.

The transformation classes for both the relational-to-attribute-based and the hierarchical-to-attribute-based data-model transformations are given in Fig. 4. Notice that we have omitted the word keyword, in order to conserve space. Additionally, each parenthesized category may have one of two possible superscripts. The first superscript "*" denotes that zero or more keywords of that category may be present in the transformation class. The second superscript "+" denotes that one or more key-
words of that category may be present in the transformation class.

In Fig. 4, the sets \( R = \{ R_{TC1}, R_{TC2} \} \) and \( H = \{ H_{TC1}, H_{TC2} \} \) are used to represent the relational and hierarchical data-model transformations, respectively. The first relational class, \( R_{TC1} \), is for relations without keys, and consists of a set of one type-name-inclusion keyword (for the relation name) and one or more elementary keywords (for the attributes of the relation). The second relational class, \( R_{TC2} \), is utilized when a relation has a key. This class consists of a set of one type-name-inclusion keyword (for the relation name), one identification keyword (for the attribute that serves as the key of the relation), and zero or more elementary keywords (for the remaining attributes of the relation). We notice that neither of the two relational classes contain referential keywords.

As noted previously, the relational model cannot support the parent-child, one-to-many relationships that are supported by the hierarchical model. So, the lack of this type of data relationship becomes apparent in the specification of the transformation classes for the relational case.

The first hierarchical class, \( H_{TC1} \), is for the root of the hierarchy and consists of a set of one type-name-inclusion keyword (for the root segment-type name), one identification-ordering keyword (for the root sequence field), and zero or more elementary keywords (for the fields of the root segment). There may be no elementary keywords for the root segment, if in the specification of the segment type only a sequence field for the root is defined. The second hierarchical class, \( H_{TC2} \), is for all of the other descendant segments in the hierarchy, and consists of a set of one type-name-inclusion keyword (for the descendant segment name), one identification-ordering keyword (for the sequence field of the descendant segment), one or more referential keywords (for the cascaded sequence fields), and zero or more elementary keywords (for the fields of the descendant segment). Furthermore, the descendant segment may also have the sequence field as the only field in the segment. With respect to our example transformation in Fig. 3, the transformation class \( H_{TC1} \) captures the Course file, while the transformation class \( H_{TC2} \) captures both the Prereq and Offering files. For instance, we see that the Course file being composed of a type-name-inclusion keyword (i.e., \(<\text{File, Course } >\) ), an identification-ordering keyword (i.e., \(<\text{Course#, i-value } >\) ), and two elementary keywords (i.e., \(<\text{Title, string } >\) and \(<\text{Descrip, string } >\) ) is exactly the class \( H_{TC1} \) shown in Fig. 4. Likewise, both the Prereq and Offering files being composed of a type-name-inclusion keyword, a referential keyword, an identification-ordering keyword and elementary keywords are exactly captured by the class \( H_{TC2} \) in Fig. 4.

Using our classification scheme, we can specify a set of transformation classes for each of the four transformations. In addition to the two sets defined in Fig. 4, we also have sets for the network and functional transformations, i.e., \( N \) and \( F \). What use are these sets of transformation classes? We are mainly concerned with determining the containment properties of these transformation classes. Is \( R \subset H \)? Is \( H \subset N \)? Are \( N \subset F \) and \( F \subset N \), making the two sets of transformation classes equivalent? The symbol \( \subset \) denotes that the data semantics of one data model are subsumed by the data semantics of another data model. From an intuitive perspective, a particular subsumption of one data model, say, the relational model, by another data model, say, the hierarchical model, implies that the hierarchical model has as least as many data constructs as the relational model. If the subsumption is not true in the other direction, then the hierarchical model also has some data constructs that are not in the relational model. What this is saying is that the hierarchical model has more data-model semantics than the relational model.

In Fig. 4, by inspection the reader can see that we would show that the relational model is subsumed by the hierarchical model by showing that for all possible relation types \( \{ R_{TC1}, R_{TC2} \} \), that \( R_{TC1} \subset H_{TC1} \) and \( R_{TC2} \subset H_{TC2} \). On the other hand, we could also show that the hierarchical model is not subsumed by the relational model, since the class \( H_{TC1} \) cannot be shown to be a subset of either \( R_{TC1} \) or \( R_{TC2} \), i.e., neither of the two relational classes contain identification-ordering keywords, which would be required in order to prove the subsumption. The complete details on all of the various subsumptions that we have proven and disproven for the four data models in our study are documented in Demurjian [15].

The questions posed above, represent only a few of the possible questions to be answered concerning the semantic containment properties of the sets of transformation classes. Overall, there are a number of different properties to be checked, namely, proper subset containment, improper subset containment, bidirectional containment and noncontainment. The end result of the application of the paradigm is a ranking of the data models. This ranking is based on the examination and comparison of the sets of transformation classes for the four data models.

In fact, we have established not one, but two, rankings of the four data models. The first ranking has

\[ \text{relational} \subset \text{hierarchical} \subset \text{network}, \text{and} \]

\[ \text{relational} \subset \text{functional}. \]

The first ranking informs us that there is no way to have either the hierarchical or the network models to be contained in the functional model, since both the hierarchical and the network models require us to maintain an ordering of the parent-child or network-set relationships.
The first ranking of data models, is also a total ordering on the set of four data models, with respect to the $\subset$ relation. It is interesting to note that in the above ranking of data-model semantics, this total ordering of set inclusion $\subset$ is equivalent to the less-than relation $<$ in the ordering of the data-model transformations in Section III-A. These two orderings do not exactly coincide, since there is no way to show that the functional model is a proper subset of either the hierarchical or the network models. Nevertheless, we find that the quantitative metrics do indeed reflect the semantic complexity of the different data models. So, we have verified that, the metrics can be used to arrive at a preliminary conclusion concerning the semantic relationships of data models. Minimally, the metrics can be used to point us in the right direction, when we begin to examine the containment properties of the data-model transformations. We have also gained a great deal of confidence in our set-theoretic analysis of the data-model semantics via data-model transformations, since we do have a metric that supports our results. Thus, we feel that we have shown that there is a very strong correlation between the study of data-model transformations and the study of data-model semantics.

In our second ranking, we redefine the transformation classes for the hierarchical and network transformations, to remove the sequencing property of each model, resulting in relational $\subset$ hierarchical $\subset$ network $\subset$ functional.

The second ranking informs us that we can achieve a different ordering at the cost of eliminating the sequencing properties of the hierarchical and the network models. This ranking now represents a pure comparison of the data models, since we have removed an aspect from both the hierarchical and network models (i.e., sequencing) that does not play a role in defining and maintaining the parent-child and network-set relationships of these two models.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we have demonstrated that the data-model semantics can indeed be measured and analyzed via the data-model transformations. We have presented two different techniques for measuring and analyzing the data-model semantics. The first technique is a quantitative metric of the data-model-transformation process and has been used to rank the different data models on the basis of the acquired transformational keywords which are required to perform a transformation. The second technique is a qualitative method and has been used to analyze the data-model semantics via the containment properties of the data-model transformation classes. The second technique also provides a ranking of the four different data models. We have found that the qualitative-method results coincide with the quantitative metric for ordering data models via their semantics. Both of these techniques make use of an experimental system, the multilingual database system, that has been designed and implemented to support many different data models (and their corresponding data languages) via the processes of data-model transformation and data-language translation.

In this conclusion, we also briefly mention our data-language work, to provide a complete perspective on our work. Like our data-model transformation work, our data-language translation efforts have also focused on developing techniques to compare and contrast data languages via their translations to a common data language. We have formalized the four data-language translations by developing sets of translation classes to represent each translation. A translation class is used to characterize the features of the source operation (i.e., an operation in SQL, DL/I, CODASYL-DML, and Daplex) and the target operation (i.e., an operation in ADL).

We have defined four source features and two target features to represent the translation from a source operation to one or more target operations. The source features are for the type of data relationship used, whether ordering is required, the data-access granularity (i.e., instance-at-a-time versus set-oriented), and the operation currency (i.e., the history sensitivity of an operation). The target features are for the number of target operations that are generated in the translation (i.e., none versus single versus multiple) and where the processing of the operation occurs in MLDS (i.e., KDS versus KC versus KC-OR-KDS).

Using these source and target features, we are able to completely characterize each data-language translation, by the formulation of a set of hexatuples, where each hexatuple represents a translation class of four source-feature values and two target-feature values. The translation classification scheme provides a generic method for specifying and comparing translations in order to demonstrate how the database semantics are preserved during the translation and execution processes. With our translation classes in hand, we are able to do a feature-by-feature analysis and comparison of our four data-language translations.

In this analysis, we have found that there is no clear dividing line between procedurality and nonprocedurality in data languages. From a procedurality standpoint, we have found that DL/I is "highly" procedural, CODASYL-DML is "moderately" procedural, Daplex is only "partially" procedural, while even SQL exhibits a "presence of" proceduralism. Conversely, we have found that SQL is "highly" nonprocedural, Daplex is "moderately" nonprocedural, CODASYL-DML is "slightly" nonprocedural, while DL/I exhibits a "presence of" nonprocedurality. Our analysis is continuing in an effort to rank the data languages on the basis of other features and characteristics. For the interested reader, we note that a number of papers based on the material presented in Section III (i.e., the quantitative metric and the qualitative method for data-model comparison) as well as a technique to analyze and compare data languages via data-language translation, all based on the work presented in Demurjian [15], are in preparation.
As to future work, one of our efforts is on continuing to use MLDS as a test-bed for developing new models, languages, and applications. The Laboratory for Database Systems Research at the Naval Postgraduate School in Monterey, CA, plans on continuing to use MLDS to develop new data models, languages, and applications. In particular, the future work includes the experimentation on temporal data models and languages that can be used to support real-time database systems and their applications and on multimedium data models and data languages that can be used to support systems and applications for different database media.

Another major, new research effort is on determining if we can provide access to a database by way of different language interfaces? For instance, can we allow the user of the Daplex language interface to access a database created by a CODASYL-DML language interface, i.e., to access a network database with Daplex? Presently, when a user of the relational/SQL language interface creates a database, the database is only accessible via SQL or ABDL. Similarly, hierarchical databases are only accessible via DL/I or ABDL, network databases are accessible via CODASYL-DML or ABDL and functional databases are accessible via Daplex or ABDL. We believe we have a system that would remove these restrictions. The implications of such a system are profound. By allowing the databases based on other data models to be accessed by data languages based on different data models, we extend our multilingual database system to a multimodel database system as well. The work of Rodeck [30] and Coker [12] indicates that a truly multilingual-and-multimodel database system is in sight.

REFERENCES

Steven A. Demurjian (M'87) received the B.A. degree in mathematics from Boston College, Chestnut Hill, MA, in 1980, the M.S. degree in computer science from Michigan State University, East Lansing, in 1982, and the Ph.D. degree in computer and information science from the Ohio State University, Columbus, in 1987. Presently, he is an Assistant Professor of Computer Science and Engineering at the University of Connecticut, Storrs. Previously, while working on his doctorate, he was an adjunct research instructor at the Naval Postgraduate School, Monterey, CA, from 1983 through 1987. He is a coauthor of a new monograph, Design Analysis and Performance Evaluation Methodologies for Database Computers, published by Prentice-Hall in 1987. His research interests include database-system performance evaluation, data models and languages, and database support for software engineering environments.

Dr. Demurjian is a member of the Association for Computing Machinery and the IEEE Computer Society.

David K. Hsiao (M'68-SM'77) received the B.A. and M.S. degrees, both in mathematics, from Miami University, Oxford, OH, in 1961 and 1964, respectively, and the Ph.D. degree in computer and information sciences from the University of Pennsylvania, Philadelphia, in 1968. Presently, he is a Professor of Computer Science at the Naval Postgraduate School (NPS), Monterey, CA, where he was Department Chairman from 1982 through 1984. Previously, he was a member of the faculty at the Department of Computer and Information Science at the Ohio State University from 1972 through 1982, a Staff and Research Engineer with Honeywell Information Science Research Center and Honeywell Information Systems, Inc., in 1971, and a member of the faculty of the University of Pennsylvania from 1968 through 1970. His research and teaching interests are in the areas of data security, database-software-system architectures, database-system performance evaluation, and database-machine organization. He is the author or coauthor of three books and over sixty research articles.

Dr. Hsiao was founder and first editor of the ACM Transactions on Database Systems, was elected Member-at-Large of the ACM Council, founded and chaired the first Technical Committee on Database Engineering, served as Governing Board member of IEEE Computer Society, and cofounded the nonprofit organization that governs the International Conferences on Very Large Databases (VLDB Conferences). He is a member of the Association for Computing Machinery and a senior member of the IEEE Computer Society.