December 1983

Features of Languages for the Development of Information Systems at the Conceptual Level

Alexander Borgida
Department of Computer Science
Rutgers University

A computer system which stores, retrieves and manipulates information about some portion of the real world can be viewed as a *model* of that domain of discourse. There has been considerable research recently on languages which allow one to capture more of the *semantics* of the real world in these computerized Information Systems -- research which has variously been labelled as Semantic Data Modeling, Semantic Modeling or Conceptual Modeling. This review paper presents a list of the features which appear to distinguish these languages from those traditionally used to describe and develop database-intensive applications, and considers the motivation for these features as well as the potential advantages to be gained through their use. The paper, which is intended for those familiar with current data processing practices, also compares in greater detail four programming languages which incorporate semantic modeling facilities, and discusses some of the methodologies and tools for Information System development based on these languages.

This work has been partially supported by the National Science Foundation under Grant No. MCS-82-10193.

A later version of this material appeared in {\it IEEE Software} Vol.2, No.1, January 1985, pp.63--73.

1 Introduction

The term "Information System" (and the acronym IS) is used in this paper to refer to any computer system which is intended to assist a class of users in the task of maintaining and accessing a pool of information on some aspect of a "real world" domain. We follow tradition here by restricting our attention to "formatted data", in contradistinction with free natural language text, pictures or continuous signals.

The view that in an IS

the symbol system in the computer is a model of the real world

has been noted and examined by numerous researchers (e.g., [Abrial 74], [Biller 77], [Langefors 77], [Tsichritzis & Lochovsky 81]) and much of the work on "conceptual modeling" (abbreviated CM henceforth) takes this as a fundamental tenet. It is important to note that the IS must model the **user's conceptualization** of the application domain, not the designer's independent perception, nor should it be a model of the way data is stored in the computer.

Given this assumption, what ought to be in a conceptual model of some universe of discourse? Adopting a rather naive mentalistic philosophy, the mental world is often seen as populated by conceptual *objects/entities*, which have associated descriptions and are *inter-related* in meaningful ways; furthermore, *activities* occur over time, with resulting changes in inter-relations; all of these are subject to *constraints*, which define the terms and/or distinguish "reality" from other possible worlds (e.g., a person can have only two parents).

In this paper we wish to survey several languages which purport to allow the description of an IS in a manner which models the real-world enterprise more *naturally* and *directly* than has been the case traditionally. The goal of this approach is to facilitate

- the **design and maintenance** of the IS, by adopting a *vocabulary* which is more appropriate for the problem domain, and by *structuring* the IS description as well as the description process;
- the **use** of the IS, by making it easier for the user to *interpret* the data stored, and thus obtain information.

The remainder of the paper is structured as follows: We first summarize some problems with traditional IS development languages in Section 2. Then, in Sections 3 and 4, we present some of the facilities for modeling the static and dynamic aspects of an enterprise which distinguish Conceptual Modeling Languages (CMLs henceforth). Finally, we consider some aspects of novel methodologies for IS development which are based on the use of CMLs, as well as computer tools supporting them.

There is currently an extensive and growing literature on "semantic data models", often concerned only with modeling the information to be stored in the database. Since we cannot hope to give appropriate attention to all of them in this paper, and since their semantics are notoriously vague, we have chosen to explicitly consider mainly those CMs which have been incorporated into *programming languages*, and thus are more likely to be properly defined and support the full description of an IS, including its activities. The specific languages to be discussed will be ADAPLEX [Adaplex 83], (based on the Semantic Hierarchy Model [Smith & Smith 77] and Daplex [Shipman 81]), DIAL [Hammer & Berkowitz 80] (based on SDM [McLeod 78]), GALILEO ([Albano et al 83], [Albano 83]) and TAXIS ([Mylopoulos et al 80], [Wong 81]). In addition, we will occasionally refer to some conceptual models which are important for historic reasons, such as Abrial's *Binary Model* ([Abrial 74]), Chen's *Entity-Relationship* model ([Chen 76]), and

Codd's extension to the relational model *RM/T* ([Codd 79]).

The appendix of this paper contains a brief summary of the fundamental features of each language together with an example which has been expressed in each of them. The example was meant to illustrate the special features of these languages and their relative strengths and weaknesses. The appendix concludes with a comparison of the 4 languages along several dimensions of possible interest.

2 The traditional approach.

Consider first how information about the objects and relationships in a conceptual model of some world appears in "traditional" information systems. The data base management systems developed commercially in the last decades have emphasized the efficient processing of large quantities of data ordinarily associated with business applications. Based on the nature of the data structuring and operation facilities that they offer, DBMS are often categorized in one of three "classical data models": hierarchic, network or relational.[1] The notion of "record" -- fixed sequence of named field values conforming to one of several record schemata -- underlies most of the currently available DBMS in a fundamental way; the hierarchic and network approaches utilize in addition the notion of link or pointer. One is therefore left with the notions of "record", "field", "co-occurrence of fields in a record" and "link between records" to encode all entities and inter-relationships of semantic interest, while the "schema" or record type definition provides the restrictions on potential relationships. So called "semantic integrity constraints" are needed to check any further conditions, and these are not available in many commercial systems.

Kent has presented detailed and cogent criticisms of the suitability of record-based information models ([Kent 79]). The list of problems includes

- the sparsity of basic constructs, described above, leads to their "overloading", since they are needed to represent a great variety of relationships and entities, and as a result these basic constructs have very weak semantics and allow information to be encoded in many alternative ways;
- although originally there appear to be clear uniform patterns for information in a particular world, many *"inhomogeneities"* arise after further consideration, and these lead to difficulties in designing proper record formats;
- records are frequently required to have "key fields" for unique identification by users, and these are often problematic in real-life; this problem is further aggravated by the practice of stating relationships between the identifiers of entities, rather than entities themselves;
- in many, though not all, record-based DBMS the description of the database does not end up being accessible to users in a manner uniform to record-access.

We will return to some of these points later in the paper, when we consider how CMs attempt to resolve such problems.

As for the modeling of the "dynamic" aspects of an enterprise, this has traditionally been accomplished by writing application programs in standard data-processing languages such as Cobol or PL/I, incorporating calls to the data-manipulation language of the DBMS which takes care of data storage. This approach itself has a number of drawbacks, including:

---

[1]The reader is assumed to be familiar with these notions (see, for example, [Date 81] or [Tsichritzis & Lochovsky 81]); also note that this use of the word "model" is distinct from the sense of modeling noted before.

- except for Cobol, these are general purpose programming languages, which have not been designed with the needs of IS modeling in mind, and hence provide few features which would facilitate this task;
- the data type structure of the host programming language is not integrated with that of the DBMS, thus requiring extensive translation between the run-time representations and storage representations, which leads to problems of inefficiency and inconsistency.[2]

Languages such as RIGEL ([Rowe 79]), PASCAL/R ([Schmidt & Mall 80]), and PLAIN ([Wasserman et al 81]) have attempted to integrate relational database manipulation and maintenance facilities into a more or less traditional programming language; however, since their persistent data types are based on the relational model, they are largely subject to the problems noted above.

3 Semantic Modeling of Objects/Entities and Associations/Relationships

The following is a list of features which in our view distinguish the approach adopted by CMLs from that of the traditional data models. It must be emphasized that this list is a collage of the features offered by many individual CMLs, but that no language has all these features.

Objects in the model correspond to entities.

A fundamental principle of work in CM is the need for a *natural bijective* mapping between the entities in the enterprise and the objects in the model. This means that the same entity is not represented by more than one thing in the model (as would be the case if there is a record representing a person as a student, and a separate record representing the same person as an employee); furthermore, the same object in the model does not represent more than one entity (e.g., a record which has the information describing both the student and his/her advisor). Problems such as the "insertion and delition anomalies" in the relational model[3] are largely due to the violation of this principle.

Distinction between reference and object

In some record-based models, such as the relational one, an object, represented by a record, cannot exist without having a corresponding visible "name" -- the primary key. In CMs, objects may exist without having "proper names", and yet be distinct from other objects. For example, in an IS supporting a library, there could be entities corresponding to authors but one would normally maintain only the name of an author, which may not uniquely identify him or her; yet, a librarian would like to be able to retrieve the books written by the author of "Waverly", without mixing in books written by other authors of the same name.

In addition, this avoids the problems inherent with unique external identifiers, such as their potential non-uniqueness, synonymy, scope and mutability (see [Kent 79], [Codd 79]), and allows partial knowledge to be recorded much more naturally (e.g., this student is a new borrower but doesn't remember his student number).

Relations encode associations between entities

Generally, mathematical relations are used to model the inter-relationships of entities in the world. An important consequence of the previous point is that relations are asserted between objects, not their names or descriptions. Thus, to represent the fact that Pierre borrowed a particular copy of a book, the Loaned-To relationship relates the corresponding entities in the IS, rather than relating Pierre's student number and the book's call-number. This helps to resolve the problem of "dangling references", which arises in record-based systems when the name in a relationship no longer refers to an entity due to mutation of the name or removal of the entity. The above three points appear to be fundamental to all CMLs, and were advocated in the database context in the pioneering work of

---

[2]For further details, consult the work of Atkinson (e.g., [Atkinson et al 82]), who has considered in depth the problem of *data persistence*.

[3]See [Date 81] for a review.

Abrial and Chen.

Three special semantic relations: attribute, type, subtype

As observed by McLeod and Smith in [Brodie & Zilles 81], many CMs single out three binary semantic relations for special treatment: *has-attribute, has-type/has-instance, has-subtype*.

An **attribute** is a functional relation of the object to which it applies, and the collection of the attributes of an object together with their values form a description of the object. For example, among the attributes of a specific book in a library, we can include a Title, whose value may be "Waverly", a Publisher, Oxford Press, and a Call Number, PR 5322 W4.

There are usually two significant differences between a library book having as one of its attributes *Borrower* say, and explicitly establishing the binary relation LOANED_TO between books and people: (i) in the former case the relation is usually asymmetric: from book to person, while in the second case it is bidirectional, and (ii) only in the second case can the relation have an attribute of its own, specifying for example the date when the book was borrowed.

A particular book, such as Waverly, is related to the object BOOKS, representing the generic concept of book or the **class** of books, by the *instance-of* relationship. If, as usual, every object is required to be the instance of at least one class, one obtains a very important typing mechanism. Through it, one can insist, for example, that all instances have only certain attributes and impose constraints on their values. Thus, all BOOKS can have a Title, which must be a string, a publisher, which must belong to the class of known publishers, etc. In most CMs, one object is allowed to be an instance of several classes, and relations are also typed.

Finally, classes are themselves related among each other by the *subclass* relationship, as between SHORT_TERM_LOANS and BOOKS, or COMPUTER_BOOKS and BOOKS. This is particularly useful when there are many classes, since the resulting hierarchy, often referred to as the *IS-A hierarchy*, *organizes* the class descriptions and often *eliminates* unnecessary duplication: if COMPUTER_BOOKS form a subclass of BOOKS, then there is no need to repeat for COMPUTER_BOOKS all the things mentioned in the definition of BOOKS, since these are automatically true of the subclass. This *inheritance* of attributes and constraints from a class to its specializations appears to be a useful abreviatory device.

The CMs provide special notation for these three relations for a number of reasons:

- by reason of their frequent occurrence in modeling;
- because they turn out to form the bases of important *axes for the organization* of large descriptions (see section 5);
- because they receive special interpretation through which certain semantically important consistencies can be expressed as part of all models. Thus, relationships can only relate existing objects (viz. "existence dependencies" in the relational model), instances of a subclass are required to be instances of all superclasses of that class, and attributes defined on one type must also be defined for all subtypes.[4].

Classes are objects themselves.

Not only can one attach definitional information to classes, but in some cases (TAXIS, SDM) classes themselves are objects, and hence may have their own attributes. This provides a natural repository for summary information (e.g., NumberOfItemsOnLoan and AverageDurationOfLoan for various subclasses of books), and "meta data" (e.g., the Publisher of an instance of COMPUTER_BOOKS must belong to the class of SCIENCE_PUBLISHERS). The latter allows for *uniform* access to the intensional part of an IS (i.e., its description) and its extensional part (i.e., the facts currently stored in it).

Support of multi-valued attributes

Many CMs recognize the utility of multi-valued attributes (e.g., a book can have several authors) and, following the idea of a functional query language introduced in [Buneman & Frankel 79], allow

[4]See [Mylopoulos & Wong 80] and [Borgida & Wong 81] for such rules

attributes to have as value sets or sequences of entities. In such cases, the CMLs provide set operators to manipulate them, and often build in conventions which greatly facilitate the concise expression of iterative procedures.

For example, in ADAPLEX or DIAL the expression *Name(Author(y))* would evaluate to the name of the author of y, if there is only one author, but would result in the set of names of the authors of book y, if there is more than one author. This is accomplished by the convention that f(x) is equal to $\{f(z) \mid z \in x\}$ if x is a set, but f(x) otherwise.

Redundancy and computed information

CMLs allow for the definition of computationaly derived information, such as derived attributes or derived classes. For example, the class of OverdueLoans is the subclass of Loans whose DueDate is less than $ToDay -- a system maintained variable --, and the Fine attribute of an OverdueLoan is the OverDueFineRate times the difference between $ToDay and the DueDate of the loan.[5]

This facility is related to the assumption held by developers of CMLs that redundancy is conceptually useful and important in (a) accommodating multiple viewpoints, and (b) reaching a complete and consistent description of a domain during IS development. Another important goal is to allow the user to treat both stored and derived information in a syntactically uniform manner, thus achieving further storage independence, since the decision to store or derive data can often be based on efficiency considerations.

Special syntax for many constraints

As noted above, one often needs to express constraints on the possible inter-relations of objects in the definition of types appearing in the model. We list below some of the special mechanisms provided for this purpose in CMs.[6]

- Domains and ranges for attributes and relations: in all CMs, the definition of a class or a relation provides for each attribute a class to which its value must belong. These "domain constraints" go far beyond the conditions on the numeric or string format of the data which are provided by most traditional DBMS.

- Cardinality of relations: restrictions on the objects y that can be related to some object x by a relation R(x,y). For example, in the Bianry Model one can express a lower and upper bound on the number of entities which can be related to a single value by a relation R and its inverse R^{-1}; thus *Parents* is a (2,2) mapping, while its inverse is $(0,\infty)$ indicating that a person must have exactly 2 parents and may have 0 or more children.

- Identifying attributes: one or more attributes may be marked as unique identifiers, or some equivalent term, denoting the fact that each object must have a distinct combination of values for these attributes;

- Optional vs. required attributes, mutable vs. immutable attributes: some attributes must have known values at all times (e.g., number of books on loan), while others may be null; also, some attributes, once assigned a value, cannot normally be changed (e.g., book title).

- Constraints on the sets of instances of classes: since an entity may belong to more than one class, some CMLs such as ADAPLEX and GALILEO, allow the designer to specify explicitly whether two classes may overlap or whether some group of classes partitions some superclass.

  In order to state general constraints, for which no special syntactic form has been made available, ADAPLEX allows general integrity constraints to be stated in a language with quantification.

---

[5]Such derived information can be defined using the view mechanism of certain relational systems such as System R.

[6]Some of these constraints can be expressed in one or another of the traditional DBMS.

4 Modeling dynamic aspects of the enterprise in CMLs

Researchers in the field of CMs have recently emphasized the need to model the activities of the enterprise as an **integral** part of the conceptual model, claiming that this information is important in the final definition of the IS.[7] As with data, it is important to capture in the model the semantics of the real-world activities, and to do so in a direct and natural manner.

The CM programming languages offer primitive data manipulations facilities such as adding and removing objects in classes, and setting and retrieving their attribute values. In addition, they provide control structures for expressing conditional and iterative execution of statements, with special concern for manipulating groups of objects, such as all the instances of a class. These languages also offer the notion of *transaction*: a procedure which is guaranteed to leave the system in a consistent state, and facilities for raising and handling *exceptions*. These can be used as a mechanism for concentrating on normal cases in transaction development, leaving special abnormal cases to be treated in a separate pass.

Information about the semantics of activities may appear in a number of forms in a conceptual model:

Description of events.

In CMs such as the original SDM ([McLeod 78]) or RM/T ([Codd 79]), objects can be classified to be *"event types"*. In RM/T, events are those entities which have associated a time of occurrence or initiation/termination. RM/T allows special semantic relations, such as "**must be followed by**" and "**may be followed by one of**" between event types, thus capturing the semantics of real-world event sequences (e.g., lending a book must be followed by the return of that book) and possibly constraining the modifications of event objects.

Encapsulation of primitive operations on a class.

In all languages, one can define special procedures for manipulating objects in semantically meaningful ways (e.g., LOANS are created by BORROWING). However, one would often like to prevent others from manipulating these objects through the primitive update operators such as "**insert** c **in** LOANS", since the designer-specified procedures can maintain effectively semantic constraints that have not been expressed declaratively. Languages such as GALILEO, DIAL and especially the Binary Model, allow one to define and enforce the notion of a class as an abstract data type, where the primitive updaters can be selectively hidden behind a list of publicly available procedures. Such a procedural definition of the underlying semantics of a class or relation is extremely powerful, from the point of view of the designer, since it offers the expressive power of the programming language together with efficiency in execution, while at the same time being transparent to the user. This transparency is in fact one its defects, since such constraints cannot be easily examined or changed by the user. For this reason, some CMs strive for a declarative way of specifying the constraints checked by these operators.

Modeling of events through transactions.

Transactions, as groups of primitive operations, are usually used in CMs to model complex activities in the world. Among others, [De Antonellis & Zonta 81], [Brodie 81] and [King & McLeod 83] have investigated the kind of information that needs to be captured in such activity descriptions.

One of the language design goals for some CMLs is to make event descriptions less "procedural" than in traditional programming languages, by taking advantage of the restricted nature of the application programs: Information Systems. For example, DIAL encourages the use of "bulk"/"aggregate" operators on successive temporary sets of objects[8] rather than the use of explicit loops through objects.

---

[7] [De Antonellis & Zonta 81] provides a review of this problem area, in addition to novel work on the subject.

[8] which can be temporarily augmented by additional attributes

A second concern of some CMLs is with *uniformity*: making the description of activities similar to those of data objects. In TAXIS, the parameters, preconditions and statements of a transaction are considered to be its attributes. This allows transactions to be viewed as entities which belong to classes, and provides for IS-A hierarchies of classes with inheritance for the description of activities. As a result, transaction descriptions to be specialized by following the specialization hierarchies of its parameters, and each specialization may add or strengthen one of the preconditions of the more general transaction.

Persistent Events.

Transactions are usually considered to be atomic activities corresponding to "instantaneous" events. However, for many commercial applications, especially in the field of office automation, one needs to specify events that last for a longer period of time. For example, once a person has borrowed a book, he may return it or renew the loan before the due date, but the number of renewals is often limited. After the due date passes, a reminder notice is sent out, and, eventually the borrowing privileges of the borrower may be withdrawn. Clearly such activities have significant semantics, such as the dynamic or temporal integrity constraints which they incorporate. Traditionally, this information is represented in a fragmented manner in disparate database facts and procedures, if captured at all. Although such a description could be programmed as a collection of concurrent events in an appropriate programming languages such as ADA, such languages lack the data management facilities needed to effectively maintain very large numbers of processes (e.g., the number of loans of a library) and to allow users to query them (e.g., how many loans have been renewed twice?).

The pioneering work of Zisman ([Zisman 77]) introduced the notion of Augmented Petri Nets to model such activities in the context of office systems. These are essentially transition-networks of potential activities, where transitions usually are caused by the occurrence of external events, the passage of time and/or receipt of internal messages. This notion was incorporated into the TAXIS framework under the notion of *scripts* ([Barron 82]), which are used to model both persistent events and user interaction with the system. The models in INCOD-DTE ([De Antonellis & Zonta 81], [Atzeni et al. 82]) and the Event Model ([McLeod & King 83], [King & McLeod 83]) also provide state-transition models of event sequencing.

5 IS design methodologies and Conceptual Models

One of the motivations for CMs was the need to capture more of the semantics of enterprise in order to serve as a requirements specification for later stages of database design (see [Lum et al 78]). The task of developing large, detailed conceptual models of enterprises which are accurate, consistent and complete, has run into the same complexity problems as those encountered in developing large pieces of general software. The solution in general appears to be to follow some consistent, systematic process for considering the myriads of details. Recent research ([Smith & Smith 78], [Borgida et al. 84a], [Brodie & Ridjanovic 83], [King & McLeod 83], [Atzeni et al. 82]) suggests that the constructs offered by CMs turn out to provide an important basis for methodologies of modeling enterprises.

At least one reason for this is that the special semantic relations noted in Section 2 are intimately associated with so-called "abstraction principles". An abstraction principle selects certain details of a description as being important and relegates the rest to a later stage of an iterative refinement process. Thus, *instance-of* corresponds to the *classification* abstraction, whereby common characteristics shared by a collection of entities are captured in the description of one entity, the class. The *aggregation* abstraction allows one to consider an entity without always having to consider it as just the sum of its parts, and is supported by the *attribute-of* relations. Finally, common characteristics of several class are factored out by the *generalization* abstraction, which of course corresponds to *subtype of*. The significance of these abstraction principles for database design was first demonstrated by the Smiths ([Smith & Smith 77]), and has since been extended and exploited by a number of researchers.

As an example, consider how TAXIS uses *specialization*, the refinement process corresponding to generalization, to build the portion of an IS for a hospital dealing with admitting patients ([Borgida et al. 84b], [Borgida et al. 84a]). The most general classes, PATIENT, DOCTOR, WARD, are described first; then successively smaller subclasses of each (e.g., OUT-PATIENT, HOSPITALIZED-PATIENT, DIAGNOSED-PATIENT,..., CARDIOLOGIST, SURGEON, CARDIAC-SURGEON,..., SURGERY-WARD, CCU, ICU,...) are described by providing additional details until all the types of entities which are expected to be encountered have been described. As in [Smith & Smith 77], attributes of more general classes are not restated for subclasses, only the differences and new attributes. One then describes the most general version of the ADMIT(doctor, patient, ward) transaction -- the parts applicable to all patients, admitting doctors and wards. The hierarchy of entity classes corresponding to the parameters of the transaction is then used to organize the description of the transaction into more and more specialized versions e.g., ADMIT(, CARDIOLOGIST,) and ADMIT(,SURGEON,) are described first, then ADMIT(, CARDIAC-SURGEON,), etc. The resulting transaction could have been described as one large monolithic procedure with many conditional statements describing special cases; the specialization process however decomposed and structured the design process. Exception handlers and scripts are also specified by descending the IS-A hierarchy of classes.

In a similar vein, in [Brodie 81] the aggregate decomposition of objects is used to systematically consider the decomposition of the actions which affect the object. For example, when a *hotel reservation* has as parts *reservation#, hotel, room, person, arrival date, departure date*, then *insert-hotel-reservation* creates a reservation#, finds the hotel, finds and updates the room, inserts the person, requests and inserts the arrival and departure dates.

Adopting a more general view, McLeod and King ([King & McLeod 83]) suggest a high level description of the enterprise which emphasizes the data and control flow, and from which one can derive the data classes which will be needed in the IS. Such a high-level description can be properly said to be a requirements specification for an IS, rather than a program for it, since it emphasizes the capture of world information which is necessary to develop an appropriate IS. This information is however not usually sufficient to provide an implementation, because it leaves all implementation issues undecided.

Conceptual modeling at the *requirements specification* level has been advocated for general software systems (e.g., [Bubenko 80], [Wilson 79]), and researchers associated with the CMLs described in this paper have developed requirements languages which are well-suited for being implemented using the corresponding CML (e.g., [Greenspan 82], [Kunin 82]).

6 Computer aids associated with CMLs

Given the claim that CMLs can be used to facilitate the development of IS, it is natural that many of the researchers have attempted to develop automatic or semi-automatic tools to assist the IS designers. In general, these provide linguistic support, development aids and implementation aids.

The ideal computer tool supporting the use of a CML is a compiler to translate a conceptual description directly into some computer representation which provides for data manipulation efficiency at a level comparable to those in traditional application systems. Prototypes of such compilers are being developed for ADAPLEX ([Chan 82]), TAXIS ([Nixon 83]) and GALILEO, -- the ADAPLEX storage management system being the most sophisticated -- but none are yet commercially available. Among the open problems is the global optimization of the more complex consistency checks inherent in CMLs, such as

referential integrity and constraints on the ranges of attributes.

CMLs can however be used for the important tasks of prototyping and testing Information Systems, by being provided with an interpreter into some interactive language such as LISP (e.g., [O'Brien 82], [Albano & Orsini 83]) . Such an interpreter verifies the syntactic correctness of the specification, and allows it to be exercised using small amounts of data. The interpreter can be augmented to provide a more complete development environment by removing burdensome clerical tasks from the designer, by performing consistency checks on the evolving design and by guiding the designer through some of the stages of refinement. For example, the TAXIS full-screen editor ([O'Brien 82]) expands the abbreviation introduced by inheritance and checks that specialization is proceeding properly, while the DIALOGO system ([Albano & Orsini 83]) allows the description to be queried and modified on-line. Similar interactive development aids appear in [King & McLeod 83] for the Event Model, and [Atzeni et al. 82] for an augmented Entity-Relationship model.

7 Conclusions

To summarize the advantages of CMs from the users' point of view, note again that the usefulness of an IS depends on two general factors: 1) the accuracy and completeness of the information stored (i.e., how well it *models* reality), and 2) the accessibility of the information to the user. CMs facilitate user access to information by

- modeling *directly* the entities and activities in the world
- using the notion of attribute to access *directly* from one object all relevant related objects[9], rather than having to navigate storage-related access paths or computed joins
- supporting the manipulation of groups of entities through set-oriented operators, multi-valued attributes and appropriate conventions
- providing uniform access to generic as well as specific information usually found in the database

By incorporating more world semantics, CMs also help control the accuracy of the information.

From the point of view of IS developers, CMs advocate an approach which emphasizes the **modelling** of the real-world enterprise, as opposed to consideration of implementation issues, and support it by providing a vocabulary specifically suited for this purpose. Furthermore, the abstraction hierarchies supported by the CML features as well as the associated automated tools help the designer in the task of gathering, structuring and maintaining the IS description.

The literature on Conceptual Modeling for Information Systems is extensive and space limitations have restricted us to only a sample of the CMs. Among others, we have omitted references to the work of research groups led by Bubenko, Langefors, Roussopulos, Sowa, Su, Wiederhold, Yao and many others. As a small compensation, we urge the interested reader to consider the articles related to conceptual modeling in the following references: [Lum et al 78], [Brodie & Zilles 81], [Chen 80], [Tsichritzis & Lochovsky 81], [Brodie et al. 84], [McLeod & King 83].

[9] often through derived attributes

ACKNOWLEDGMENTS

Some of the ideas in this paper were first developed in co-operation with Professor John Mylopoulos, to whom I am indebted for numerous stimulating discussions. I am also grateful for the following people for commenting on various portions of this manuscript, often on short notice: Prof. A.Albano, Dr.A.Chan, Dr.R.Orsini, Dr.G.LaFue.

I. Further details of four CMLs, including an example.

I.1 An example

The following toy problem will be used to present in a more concrete manner the features of the 4 programming languages surveyed in this paper. The problem concerns a portion of an IS for supporting the activities of a library. We have selected a few specific facts to be represented in order to highlight the capabilities of the languages under consideration; for this reason, the example will be very incomplete and will ignore a number of important real-world distinctions, such as that between a book concept (what it is about, who wrote it, when was it written, etc.) and particular physical copies of that book.

For our purposes, the library has a set of books (one copy of each book), each identified by a unique "call number". Each book has a title, zero or more authors, as well as a publisher. The books are grouped according to subject matter according to a scheme specific to this library, and Computer Books form one such subclass; these books have an ACM Computing Reviews descriptor and are published only by Science Publishers.

Book authors have first and last names; the librarians are expected to use their knowledge to distinguish the authors of books at the time the book descriptions are entered into the IS, but the basis of these distinctions (e.g., biographical information in the book) should not be incorporated in the IS.

Books may be loaned to borrowers, which may be institutions such as other libraries, or persons, such as students or faculty members. Lending books to institutions requires additional processing, which will remain unspecified here.

A book can normally be borrowed for 2 weeks, and two 1-week extensions are allowed, if the extension is performed before the due date. Some books may be placed on restricted loan; these can only be taken out for 3 days and cannot be renewed. In general, any person may have on loan at most 5 books at any time, and at most one short-term loan book; this requirement may be waived at the discretion of the librarian.

I.1.1 The example in ADAPLEX

The ADAPLEX language provides a marriage of the general-purpose language ADA to the notion of classes and sets of objects, which persist even after the end of the program. Attributes are viewed as functions from objects to other objects and are also persistent. The language provides facilities for set manipulation and stating quantified formulas over sets and classes, and incorporates the kinds of notational abbreviations described in Section 3. Adaplex also provides the capability to state integrity constraints, such as the use of null values, membership in multiple classes, and key attributes; arbitrary constraints can be expressed in a version of First Order logic, although such integrity checking is currently not yet implemented.

```
type authors is
   entity
        first_name: STRING(1..20);
        last_name: STRING(1..40);
   end entity;
```

```
type publishers is ...
subtype science_publishers is publishers ...


type borrowers is ...
subtype institutions is borrowers ...
subtype persons is borrowers ...
subtype faculty is persons ...


type books is
    entity
        call#: INTEGER;
        title: STRING(1..120);
        publisher: publishers;
        written_by: set of authors;
    end entity;
unique call# within books;


subtype computer_books is books
    entity
        CRsubject_codes: set of STRING(1..40);
    end entity;

restricted_publishers: integrity  for every y in computer_books:
                                  publisher(y) is in science_publishers;



subtype short_term_loan_books is books
   entity
        expiry_of_restriction: dates;
        requester: faculty;
   end entity;


subtype regular_loan_books is books
   entity
   end entity;

overlap computer_books with  short_term_loan_books, regular_loan_books;


subtype books_on_loan is books
   entity
        loaned_to: borrowers;
        due_date: dates;
        renewals_left: INTEGER range 0..2  :=0;
   end  entity;


subtype books_available is books[10]
```

[10] In ADAPLEX, every object must belong to some "terminal class" -- i.e., one which has no subclasses of its own. For this reason, it is usually proper to define "complement classes" when defining subclasses.

```
    entity;
    end entity;


The  following  must be declared in a separate Ada package:


function overdue_books() return boolean is
    begin
         return({b in books where before(due_date(b),$today)});
    end overdue_books;


function books_borrowed(p:borrowers) return set of books is
    begin
         return({b in books_on_loan where loaned_to(b)=p});
    end books_borrowed;


procedure TakeOut(b: in books_available; p: in borrowers) is
  due_on: date;
  begin
    atomic
         if (count(books_borrowed(p)) >= 5) then raise loan_limit; end if;
         if (b is in short_term_loan_books)
             then if (p is not in persons) then raise failure; end if;
             elsif {y in books_borrowed(p) where
                           y is in short_term_loan_books} is not empty
                  then raise loan_limit; end if; end if;

         if (b is in short_term_loan_books)
              then due_on := add(3,$today);
              else due_on := add(14, $today); end if;
         move b from books_available into books_on_loan (loaned_to => p,
                                                         due_date => due_on);
         if (b is not in short_term_loan_books)
               then renewals_left(b) := 2; end if;
    end atomic;
  end TakeOut;
```

I.1.2 The example in GALILEO.

GALILEO also extends an existing programming language to allow for database maintenance and manipulation; in this case the language is ML, which is a strongly typed, functional language -- one in which functions are first class objects. One of the fundamental concepts of ML is that of "environment": basically a set of bindings of values to identifiers; environments can then be added or subtracted from each other, and new bindings can be added or dropped from an environment. GALILEO makes environments themselves first class citizens, which can hence be bound to identifiers themselves, and then uses this powerful concept to define classes, databases, views and modules. GALILEO also introduces the notion of type hierarchy, which can be used to define hierarchies of classes as in other semantic models. GALILEO achieves the most consistent integration of general programming facilities with those needed for IS applications, and is likely the most advanced of the CMLs.

```
use Library :=

rec authors class
    author <->
        (first-name: string and
        last-name: string)

and books class
    book <->
        (call#: string and
        publisher: Publisher and
        title: string and
        authors: seq Author)
        key (call#)

and  computer-books subset of books class
    computer-book <->
        (is book ext
         publisher: SciencePublisher and
         CRsubject-code: seq string)

and short-term-loan-books subset of books class
   short-term-loan-book <->
        (is book and
         expiry-of-restriction: date and
         requester: faculty)

and books-on-loan partition of books with books-available class
   book-on-loan <->
        (is book and
         loaned-to: borrower and
         due-date: var date and
         renewals-left: default var 0: var (0 or 1 or 2)

and overdue-books :=
        derived   books-on-loan with before(due-date, $today)


and borrowers class
   borrower <->
        (name: string and
         address: string and
```

```
          books-borrowed := derived (all Books-on-loan with loaned-to = this) )

and persons class ...
and institutions class ...


 TakeOut(b:book, p:borrower)   :=
        (if (count(books-borrowed of p) >= 5) then failwith loan-limit;
         if  b alsoin books-on-loan then fail;
         if  b alsoin short-term-loan-books
                then if not (p alsoin Persons) then fail
                          else if (some x in books-borrowed of p
                                       with x alsoin short-term-loan-books)
                                  then failwith  loan-limit;

        use due-on :=
                 if (b is in short-term-loan-books) then add(3,$today)
                 else add(14, $today)

        in

          inBooks-on-loan(b,b and loaned-to := p and due-date := var due-on);

        if not (b alsoin short-term-loan-books) then at renewals-left of b := 2;
        )
```

I.1.3 A Taxis description

The TAXIS language also considers data design to be of central importance, and thus adopts the fundamental view shared by all CMs: objects inter-related through properties, grouped into classes, which are arranged in an IS-A hierarchy. TAXIS however introduces meta-classes as a mechanism to describe uniformly object and meta-information, and then extends these ideas uniformly to the description of procedures (transactions), exceptions and exception handlers. Thus, for example, procedures have as properties their parameters and statements, and can be specialized to yield IS-A hierarchies of procedures.

In Taxis, one could set up a meta-class MATERIAL_TYPE, which would to record information about the different kinds of books that is uniformly applicable or summary in nature; for example, LoanDuration or the number of books in the class. The class BOOKS, and any desired subclasses such as SHORT_TERM_LOAN_BOOKS, can have different values for these attributes.

Since TAXIS permits only "downward" movement of objects in the IS-A hierarchy of classes, one cannot elegantly represent the lending and return of books through the BOOKS-ON-LOAN subclass, and we must resort to creating the class of LOANS; this does have the advantage that one can conveniently keep information about past loans, even when the books are returned.

Also, since TAXIS does not allow set-valued attributes, the relationship between authors and books must also be represented using a separate class.

```
dataclass BORROWERS with
            attribute
                name: STRINGS
                address: ADDRESSES
                #OutsdandingLoans: 0..5
        end BORROWERS;

dataclass INSTITUTIONS isa BORROWERS with ...

dataclass PERSONS isa BORROWERS with
            attributes
                hasShortLoan: {true, false}
        end PERSONS;

dataclass FACULTY isa PERSONS with ...



dataclass AUTHORS with
           characteristics
                firstName: STRING;
                lastName: STRING;
        end AUTHORS;


metaclass MATERIAL_TYPE with
            attributes
                loanDuration: INTEGERS
                numberOfItems: INTEGERS
        end MATERIAL_TYPE;
```

```
dataclass  BOOKS in MATERIAL_TYPE with
             characteristics
                 call#: INTEGERS
                 publisher: PUBLISHERS
                 title: STRING
             keys
                 bookKey: (CALL#)
         end BOOKS;


dataclass WRITTEN_BY with
            attributes
                 work: BOOKS
                 writer: AUTHORS
         end WRITTEN_BY


dataclass COMPUTER_BOOKS isa BOOKS with
             characteristics
                 publisher: SCIENCE_PUBLISHERS
                 CRSubjectCodes: STRING
         end COMPUTER_BOOKS;


dataclass SHORT_TERM_LOAN_BOOKS in MATERIAL_TYPES isa BOOKS with
             characteristics
                 expiryOfRestriction: DATES
                 requester: FACULTY
         end SHORT_TERM_LOAN_BOOKS;

    SHORT_TERM_LOAN_BOOKS.loanDuration <- 3;


dataclass REGULAR_LOAN_BOOKS in  MATERIAL_TYPES isa BOOKS;
    REGULAR_LOAN_BOOKS.loanDuration <- 14;

dataclass LOANS with
             characteristics
                 item: BOOK
                 loanedTo: BORROWER
             attributes
                 dueDate: DATE
                 renewalsLeft: 0..2
             keys
                 loanKey: (item, loanedTo)
         end LOANS;

TestDefinedClass OVERDUE_LOANS isa LOANS[11]

         attribute test on  LOANS, ANY_CLASS is CHECK_DUEDATE;

         transaction  CHECK_DUEDATE(y:LOANS, C:ANY_CLASS) isa BOOLEAN_FUNCTION
             actions
```

[11] A Test-Defined class has associated a boolean function which filters out instances of the superclass which meet certain computationally specified criteria.

```
                if C=OVERDUE_LOANS ∧ Before(y.dueDate,$Today)
                   then return(true) ;
                   else return(false);
        end;


transaction TakeOut(b:BOOKS, p:BORROWERS) with
   prerequisites
        available: begin;
                        get x from loans with item=b;
                        if x = nil then true else false;
                   end;  exc FAILURE
        notTooManyOut: (p.#OutsdandingLoans < 5) exc LOANLIMIT(who:p)
   actions
        makeLoan:   Insert x in LOANS with item=b, loanedTo=p, renewalsLeft = 2;
        setDueDate: x.dueDate <- Add($Today, minclass(b).loanDuration);
        count:      p.#OutstandingLoans <- p.#OutstandingLoans + 1;
end;


transaction TakeOut(b:SHORT_TERM_LOAN_BOOKS, p:BORROWERS) with
   prerequisites
        onlyPeople: (p instance-of PERSONS) exc FAILURE
        onlyOne:    (p.hasShortLoan = false) exc LOANLIMIT(who:p)
   actions
        resetRenewals:  p.renewalsLeft <- 0;
        setShortLoan:   p.hasShortLoan <- true;
end;
```

Other specializations of TakeOut, such as one for institutional borrowers would follow.

If storage space is at a premium, the designer may choose to compute certain attributes, such as #OutstandingLoans or hasShortLoan, rather than store them. This can be accomplished by defining, for example, the attribute

```
#OutstandingLoans: COUNT_LOANS
```

and the transaction COUNT_LOANS, which counts all instances of LOANS with this object as borrower:

```
transaction COUNT_LOANS(b:BORROWERS) is FUNCTION with
    locals
        ct:INTEGER;
    actions
        ct <- 0;
        for each instance x of  LOANS do
            if b=x.loanedTo then ct <- ct+1;
        return(ct);
end;
```

Finally, it is most likely that the lending of books would be modeled by **scripts** in TAXIS, rather than transactions and the class LOANS, since it involves long-term events, dynamic constraints and user communication. Such a script model was sketched in Section 4.

I.1.4 The example in DIAL.

The fundamental philosophy of DIAL is that a programming language for IS design should concentrate on features which allow for the succinct expression of the most frequently recurring patterns in database application programs, at the possible cost of generality, and that data description ought to be the central aspect of IS development. DIAL has at its center the conceptual modeling features of the SDM language for modeling data; the procedural aspects of an IS can be expressed through specialized and restrictive control structures, not the traditional IF-THEN-ELSE conditional and WHILE-loops.

DIAL also incorporates facilities for developing user interfaces through the notion *forms* and form-filling. This is an important concept since a great deal of business application programs is concerned with input and output of information. Forms are also viewed as objects, and the user is expected to provide values for their attrbiutes; DIAL then provides special declarative facilities to specify the syntax of the user interaction for filling each attribute (e.g., how and when to prompt for an answer, what value is expected).

```
class AUTHORS
  attributes
        first-name
                declaration: STRING
        last-name
                declaration: STRING
                mandatory

class BOOKS
   attributes
        call#
                declaration: ISBN_NUMBERS
                mandatory
        publisher
                value-class: PUBLISHERS
        title
                declaration: STRING
        written-by
                value-class: AUTHORS, multi-valued
   identifiers:
        call#


class COMPUTER_BOOKS
   derivation: subset of BOOKS
    attributes
        CRsubject-code
                declaration: SUBJECT-CODES, multi-valued

class SHORT-TERM-LOAN-BOOKS
   derivation: subset of BOOKS
   . . .


class LOANS
   attributes
        item
                value-class: BOOKS
        loaned-to
                value-class: BORROWERS
```

```
        due-date
                value-class: DATE
        renewals-left
                value-class: 0-to-2


class BORROWERS
   attributes
        books-borrowed
                derivation: Item of match to LOANS on loaned-to, multi-valued
       . . .


class PERSONS  derivation: subset of BORROWERS  . . .

class INSTITUTION derivation: subset of BORROWERS  . . .


create_proc norm.LOAN(b:BOOKS, p:BORROWERS, duration:INTEGER)[12]
   [Initattr(item,b)
    Initattr(loaned-to,p)
    Initattr(due-date, add(duration,Today()) )
    Initattr(renewals-left,2) ]

update_proc NoRenewals(loan:LOANS)
   [updateattr(loan,renewals-left,0) ]

procedure TakeOut(b:BOOKS, p:BORROWER) {transaction}
   {signals too-many, failure}
        [signal too-many if count(p.books-borrowed)>5
        type <- "short" if (b is in SHORT_TERM_LOAN_BOOKS)
        type <- "normal" if not(b is in SHORT_TERM_LOAN_BOOKS)
        signals failure if type="short" and  not (p is in PERSONS)
        signals too-many if type="short" and
                            0<count(p.books-borrowed and SHORT_TERM_LOAN_BOOKS)
        time <- 14 if type = "normal"
        time <- 3  if type = "short"
        z <- create_LOAN (b:b, p:p, duration:time)
        NoRenewals(loan:z) if type = "short"
        ]
```

[12]This procedure encapsulates the creation of LOAN instances.

I.2 A short note on approaches to implementing CMLs

Finally, it may be instructive to consider the different approaches to implementing CMLs which are currently being considered.

The ADAPLEX compiler is basically a pre-processor which separates out the portions of the program dealing with persistent data management, and passes the remainder of the code to a regular ADA compiler, which would access data through function calls. The implementors of ADAPLEX have chosen to provide special sophisticated data structuring facilities for the database, thus taking full advantage of the object-oriented nature of the CML and resulting data base, but had to avoid touching the ADA compiler, which is not yet available.

In contrast, the implementors of GALILEO are modifying an actual compiler for the language ML, on which GALILEO is based. As a result, GALILEO can provide more uniformity in the persistence of the various datatypes, since their run-time implementation is known.

In order to decrease the total effort required in writing the compiler, the implementors of TAXIS chose to translate TAXIS programs entirely into a language which provides traditional data management facilities, namely PASCAL/R. Similar translation could have been provided into PL/I plus calls to some other relational DBMS. Such an approach allows TAXIS to take advantage of the considerable work on optimization in DBMS, but at the cost of not being able to fine-tune this for an object-oriented database. A similar philosophy was adopted by Kulkarni [Kulkarni 83], who implemented an extension of the DAPLEX language in PS-Algol ([Atkinson et al 82], [Atkinson et al 83]), a language which allows for the storage of persistent data.

In conclusion, none of the languages considered above appears to be clearly superior to the others; in fact there is room for a language which would capture the best of the ideas present in each language, such as the functional treatment of attributes, especially sets, in ADAPLEX, metaclasses, procedural specialization and scripts in TAXIS, derived classes and attributes, as well as forms for information input, in DIAL, and the modularization facilities and rich but uniform type structure of GALILEO.

References

[Abrial 74] Abrial, J.R.
Data semantics.
In J.W.Klimbie and K.L.Koffeman (editors), *Data management systems*, pages 1-59. North Holland, Amsterdam, 1974.

[Adaplex 83] .
ADAPLEX: Rationale and Reference Manual.
Technical Report CCA-83-03, Computer Corporation of America, May, 1983.

[Albano 83] Albano, A., L.Cardelli and R.Orsini.
Galileo: a strongly typed, interactive conceptual language.
Technical Report 83-11271-2, Bell Laboratories, Murray Hill, N.J., July, 1983.
(To appear in ACM Transactions on Databases).

[Albano & Orsini 83]
Albano, A. and R.Orsini.
Dialogo: an interactive environment for conceptual design in Galileo.
In S.Ceri (editor), *Methodology and Tools for Database Design*, pages 229-253. North-Holland, 1983.

[Albano et al 83] Albano, A., M.Capaccioli, M.E.Occhiuto and R.Orsini.
A modularization mechanism for conceptual modeling.
In *Proc. 1983 Very Large Databases Conference*, pages 232-240. 1983.

[Atkinson et al 82] Atkinson, M.P., K.J.Chisholm & W.P.Cockshott.
PS-Algol: an Algol with a persistent heap.
ACM SIGPLAN Notices 17(7), July, 1982.

[Atkinson et al 83] Atkinson, M.P., P.J.Bailey, K.J.Chisholm, W.P.Cockshott and R.Morrison.
The Persistent Object Management System.
Technical Report, Dept. of Computer Science, University of Edinburgh, March, 1983.

[Atzeni et al. 82] Atzeni, P., C.Batini, V.de Antonellis, M.Lenzerini, Villanelli and B.Zonta.
A computer aided tool for conceptual database design.
In H.J.Schneider and A.Wasserman (editor), *Automated tools for Information System design*, pages 85-106. North Holland, 1982.

[Barron 82] Barron, J.
Dialogue and process design for Interactive Information Systems using TAXIS.
In *Proc. ACM SIGOA Conf. on Office Information Systems*, pages 12-20. June, 1982.

[Borgida & Wong 81]
Borgida, A.T. and H.K.T.Wong.
Data models and data manipulation languages: complimentary semantics and proof theory.
In *Proc. Very Large Data Bases Conference 1981*, pages 260-271. Cannes, France, September, 1981.

[Borgida et al. 84a]
Borgida, A., J.Mylopoulos and H.K.T.Wong.
Generalization as a basis for software specification.
In *[Brodie et al 83]*. , 1984.

[Borgida et al. 84b]
Borgida, A.T., J.Mylopoulos and H.K.T.Wong.
Generalization/specialization as a basis for software specification.
In *On Conceptual Modelling*[13], pages 87-114. , Amsterdam, 1984.

[Brodie 81] Brodie, M.L.
On modelling behavioural semantics of databases.
In *Proc. Very Large Data Bases Conference 1981*, pages 32-43. September, 1981.

[Brodie & Ridjanovic 83]
Brodie, M.L. and D.Ridjanovic.
On the design and specification of database transactions.
In *[Brodie et al 83*]. , 1983.

[Brodie & Zilles 81]
Brodie,M.L. and S.N.Zilles,(Editors).
Proc. ACM workshop on data abstraction, databases and conceptual modelling.
ACM SIGART/SIGMOD/SIGPLAN Notices, January 1981, Pingree Park, Colorado, 1981.

[Brodie et al. 84] Brodie, M.L., J.Mylopoulos and J.W.Schmidt.
On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages.
Springer Verlag, 1984.

[Bubenko 80] Bubenko, J.A.
Information modeling in the context of system development.
In *IFIP Congress 1980*. 1980.

[Buneman & Frankel 79]
Buneman, P. and R.E.Frankel.
FQL - a funtional query language.
In *Proc. 1979 ACM-SIGMOD Conference*, pages 52-58. June, 1979.

[Chan 82] Chan, A., S.Fox, W.K.Lin, A.Nori and D.R.Ries.
Storage and access structures to support a semantic data model.
In *Proc. 1982 Very Large Data Bases Conference*. 1982.

[Chen 76] Chen, P.P.S.
The Entity-Relationship model: towards a unified view of data.
ACM Trans. on Database Systems 1(1):9-36, March, 1976.

[Chen 80] Chen, P.P.Y. (editor).
Entity-Relationship Approach to Systems Analysis and Design.
North Holland, Amsterdam, 1980.

[Codd 79] Codd, E.F.
Extending the database relational model to capture more meaning.
ACM Trans. on Database Systems 4(4):395-434, December, 1979.

[Date 81] Date, C.J.
An Introduction to Database Systems.
Addison-Wesley, 1981.

[De Antonellis & Zonta 81]
De Antonellis, V. and B.Zonta.
Modelling events in data base applications design.
In *Proc. 1981 Very Large Data Bases Conference*, pages 23-31. 1981.

[Greenspan 82] Greenspan, S.J., A.Borgida and J.Mylopoulos.
Capturing more world knowledge in the requirements sepcification.
In *6th Int. Conf. on Software Engineering*, pages 225-233. 1982.

[Hammer & Berkowitz 80]
Hammer, M. and B.Berkowitz.
DIAL: A programming language for data intensive applications.
In *Proc. ACM-SIGMOD Conference*, pages 75-92. May , 1980.

[Kent 79] Kent, W.
Limitations of record based information models.
ACM Trans. on Database Systems 4(1):107-131, March, 1979.

[King & McLeod 83]
King, R. and D.McLeod.
The event database specification model.
In *Proc. Second Int. Conf. on Databases: Improving Useability and Responsiveness*. Jerusalem, Israel, June, 1983.

[Kulkarni 83] Kulkarni, K.G.
Extended functional data model - user manual.
Persistent Programming Research Report 7, Dept. of Computer Science, University of Edinburgh, September, 1983.

[Kunin 82] Kunin, J.
Analysis and specification of office procedures.
PhD thesis, MIT Laboratory for Computer Science, February, 1982.

[Lum et al 78] Lum, V. et al.
1978 New Orleans Data Base Design Workshop report.
Technical Report RJ2554(33154), IBM, 1978.

[McLeod 78] McLeod, D.
A semantic data base model and its associated structured user interface.
PhD thesis, Laboratory for Computer Science, Massachusets Institute of Technology, August, 1978.

[McLeod & King 83]
McLeod, D. and R.King.
Semantic database models.
In S.B.Yao (editor), *Principles of database design*. Prentice Hall, 1983.
(forthcoming).

[Mylopoulos & Wong 80]
Mylopoulos, J. and H.K.T.Wong.
Some features of the Taxis data model.
In *Proc. 1980 Very Large Data Bases Conference*. Montreal, P.Q., October, 1980.

[Mylopoulos et al 80]
Mylopoulos, J., P.A.Bernstein and H.K.T.Wong.
A language facility for designing interactive database-intensive systems.
ACM Trans. on Database Systems 5(2):185-207, June, 1980.

[Nixon 83] Nixon, B.
A Pascal/R compiler for Taxis.
Master's thesis, Dept. of Computer Science, University of Toronto, 1983.

[O'Brien 82] O'Brien, P.
A TAXIS editor.
Master's thesis, Dept. of Computer Science, University of Toronto, 1982.

[Rowe 79] Rowe, L. and K.Shoens.
Data abstraction, views and updates in RIGEL.
In *Proc. SIGMOD Conference*. ACM, May, 1979.

[Schmidt & Mall 80]
Schmidt, J. and M.Mall.
Pascal/R Report.
Technical Report No.66, Fachbereich Informatik, Hamburg University, January, 1980.

[Shipman 81] Shipman, D.W.
The functional data model and the data language DAPLEX.
ACM Trans. on Database Systems 6(1):140-173, March, 1981.

[Smith & Smith 79] Smith, J.M. and D.C.P.Smith.
A database approach to software specification.
TR CCA-79-17, Computer Corporation of America, April, 1979.

[Smith & Smith 77] Smith, J.M. and D.C.P.Smith.
Database abstractions: aggregation and generalization.
ACM Trans. on Database Systems 2(2):105-133, June, 1977.

[Smith & Smith 78] Smith, J.M. and D.C.P.Smith.
Principles of database conceptual design.
In *NYU Symposium on database design*. New York, May, 1978.

[Tsichritzis & Lochovsky 81]
Tsichritzis, D. and F.Lochovsky.
Data models.
Prentice Hall, 1981.

[Wasserman et al 81]
Wasserman, A.I., D.D.Sheretz, M.L.Kersten and R.D.van de Reit.
Revised report on the programming language PLAIN.
ACM SIGPLAN Notices 16(5), May, 1981.

[Wilson 79] Wilson, M.
A semantics-based approach to requirements analysis and system design.
In *Proceedings of IEEE COMPSAC 79*. November, 1979.

[Wong 81] Wong, H.K.T.
Design and verification of interactive information systems using TAXIS.
Technical Report TR CSRG-129, CSRG, University of Toronto, April, 1981.
forthcoming PhD thesis.

[Zisman 77] Zisman, M.
Use of production systems for modelling asynchronous parallel processes.
In D.Waterman and F. Hayes-Roth (editors), *Pattern-directed inference systems*.
Academic Press, 1977.

Table of Contents