Report Number: WUCS-80-03

1980-11-01

An Abstract Model of Unstratified Database System

Takayuki D. Kimura, Jerome R. Cox Jr., and Will D. Gillett

A semantic data model is introduced with the following capabilities: (1) Abstraction mechanisms for aggregation, generalization and classification, (2) Unstratified control of the database content, (3) Refined control of intentional and extensional information, and (4) Extensive semantic consistency checking. The basic features of the model are illustrated through a scenario of interactions between the user and the database system (using the proposed model) for constructing a simple database on technical publications.

An Abstract Model of an Unstratified Database System

T. D. Kimura, J. R. Cox, Jr., and W. D. Gillett

WUCS-80-03

November 1980

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Presented at the Fourteenth Hawaii International Conference on System Sciences, January 1981

This work was supported in part by the NCHSR under Grant HS03792 and the NIH under Grant RR00396

ADS. Section 5 presents a summary of the capabilities present in ADS.

2. AN OVERVIEW OF ADS

We consider a database system as a communication mechanism for a community of users to share their views of reality. Like a community bulletin board, the users register in the database system their judgements about reality, expecting other users to retrieve them; users also can retract old judgements. The users interact with the database system through a symbolism (or a language) whose syntax and semantics are well-defined and known to every user in the community. The depository of the symbols that represent collectively the history of user judgements is called the database. The rest of the database system that manages the content of database, we call the database manager (not a human manager). (Figure 1).

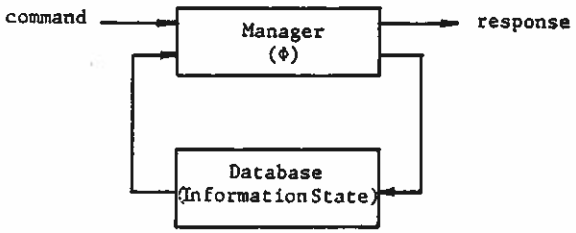

Figure 1: Database System (as a state machine)

ADS models, at a low conceptual level, a database as a collection of symbols and a database system as a language processor (like a compiler). Note again that, from our point of view, a database is a linguistic entity for which there exists associated rules of syntax and semantics. Therefore, a database design is, in a very broad sense, a language design.

Because of its primary importance in the design and usage of a database system, the language through which the users communicate their judgements and queries to the database system is called here the base language (B). The importance of the base language is highlighted by the fact that the users can communicate with each other (through the database system) only the part of reality which is describable by the base language.

Conceptually speaking, a database is a collection of expressions from B, each representing a user judgement. We call a symbolic representation of a judgement a declaration, i.e., a database is a collection of declarations.

The term 'judgement' often connotes an intelligent decision making process, but we use the term here in a broader sense to mean any process capable of extracting knowledge about reality. (One person's trivial fact may be a result of another person's deep judgement.)

2.1 CONSISTENCY CHECKING

The database manager, the manager in short, manages the information in the database. Insertion, updating and retrieval of the contents of the database is carried out only through the manager. It is important to understand that the users share (access) the knowledge in the database through the manager, but the manager does not have the capability of validating the knowledge, i.e., it cannot confirm or deny whether a particular judgement represents reality or not. This issue is entirely outside the database system capability. From this point of view, the database manager is literally a manager.

The manager, however, has the capability of enforcing the policy set by the users and registered in the database about what information should be allowed in the database and what form it must take. These policy statements must be in the database as a particular type of user judgement expressed in the language B (i.e., as declarations). There is no significant difference between policy statements enforced by the manager and other statements representing other user judgements, i.e., the policy enforced by the manager is virtually under the user's control. One of the typical policies that the manager enforces is to check the consistency of a new judgement with old ones already in the database. What types of consistency the manager should check (and to what extent) must be provided by the user. In other words, the definition of the consistency of the information in the database is the user's responsibility, but enforcing the consistency by rejecting any inconsistent judgements is the manager's responsibility.

2.2 UNSTRATIFIED CONTROL

Special attention must be paid to the fact that the specification of the rules about the contents of the database (e.g., consistency rules) are contained in the database itself as one of many other user judgements. This capability of mixing a metalanguage (general judgements or schema) and an object language (specific judgements or data) within a database provides the database system with a capability analogous to the stored-program capability of von Neumann machines. The manager's capability to interpret an expression in B either as a representation of a user judgement or as an uninterpreted symbol of B (depending upon the context), is called 'unstratified control' or 'unstratified interpretation'.[7] Obviously, humans have such a capability: consider our interpretation of the word "Smith" in:

(1) Smith is an Anglo-Saxon.

(2) Smith is an Anglo-Saxon name.

Unstratified control also occurs in certain automated systems. For example, the programming language system LISP has the same capability: an S-expression can be interpreted (by EVALQUOTE) as

AN ABSTRACT MODEL OF AN UNSTRATIFIED DATABASE SYSTEM

Takayuki D. Kimura, Jerome R. Cox, Jr. and Will Gillett
Department of Computer Science
Washington University, St. Louis, Missouri 63130

Abstract

A semantic data model is introduced with the following capabilities:

(1) Abstraction mechanisms for aggregation, generalization and classification,

(2) Unstratified control of the database content,

(3) Refined control of intensional and extensional information, and

(4) Extensive semantic consistency checking.

The basic features of the model are illustrated through a scenario of interactions between the user and the database system (using the proposed model) for constructing a simple database on technical publications.

1. INTRODUCTION

The purpose of this paper is to introduce a data model with anticipated applications to medicine. The presentation of the model will be informal, and we will present the unique capabilities of the model primarily through examples. The formal definition of the model is given elsewhere.[1]

The Abstract Database System (ADS) is a formal model of a database system that uses a common language for storing and describing data and its schemata and for communicating with the user.

ADS views its database as a linguistic entity, i.e., as a set of symbols arranged to constitute a language which is interpretable by ADS.

ADS is concise, self-defining, and supplies the capability to enforce semantic constraints, as defined by the user. These constraints can be either local (i.e., properties of a database object) or global (i.e., relationships between different database objects). Since ADS is a formal model, it is not intended to be used directly by a human user, but instead, constitutes a basic semantic model for developing secondary or vernacular models.[2]

This work was supported in part by the National Center for Health Services Research under Grant HS03792 and the National Institutes of Health under Grant RR00396.

1.1 BACKGROUND AND CAPABILITIES OF ADS

Our work on ADS has been guided by a specific application area, medical information systems.[3] In such an environment, any database must be capable of withstanding multiple technological and administrative eras. As the nation's morbidity continues to shift from acute to chronic disease (e.g., diabetes, hypertension), the duration of a patient's illness will constitute a greater fraction of the patient's lifetime. This is a major reason for our concentrating on a formal model, capable of being implemented in many different technologies.

As doctors, nurses, and other non-technical personnel (with respect to computers, at least) become more intimately associated with automated databases, the mode by which they interact with the database must be flexible enough to meet each user's need. Doctors and administrators may choose to 'view' virtually the same information in entirely different ways. Thus, there must be some mechanism to translate from one view of information to another (suppressing some information in the process.)

Since the extension of a name may be a part of the descriptor of another name, as we will discuss later, a change in the extension of one name may cause a change in the intension of another name. Therefore, an extensional judgement on one name may require a chain of consistency checking due to possible changes in the intension of other names.

This is the essence of consistency checking in ADS between extensional judgements and intensional ones. As far as the consistency is concerned, what should be checked is inate to ADS, i.e., all extensional judgements.

There is another kind of checking capability ADS provides at the level of a transaction: the user controls what should be checked based on the user's view of the world. The mechanism is an assertion. When an a-name is defined with an associated a-descriptor, there is no commitment, on the user's part, as to the logical value of the a-descriptor. That is to say, ϕ(a-descriptor) may be T at one point in time, and F at another. However, once an extensional judgement "a-name:=T" is entered, then 'ϕ(a-name)=T' must hold invariently thereafter.

The set of a-names whose extensions are defined, i.e., extensionally observed valid assertions, is called the constraint set. The validity of the constraint set is checked by ADS after each transaction, (not after each judgement). If the validity is violated by a transaction, then the transaction will not be accepted by ADS, even if all of the judgements in the transaction are acceptable to ADS. Note that a transaction is a sequence of judgements, intensional and extensional, and the constraints may be violated inside the transaction, but not after the last judgement. Also, note that the constraint set itself may be modified during the transaction.

2.12 SUMMARY

In summary, ADS consists of the interpretation rule (ϕ) and the information state. The information state is a collection of user defined names of objects along with the two attributes, intension (μ) and extension (τ), associated with each name. Namely, an information structure is a set of triplets:

(name, μ(name), τ(name)).

ADS accepts sequentially a transaction or a query. A transaction transforms the information state from one 'legal' state (i.e., a state in which the consistency and the constraints are valid) to another. A query does not change the information state.

3. A SIMPLE DATABASE ON PUBLICATIONS

The objective of this section is to describe, informally, a simple and hypothetical slice of reality dealing with publications of technical papers. This will be used in a subsequent section to help the reader better understand how ADS represents a users view of reality. First, we will describe the structure of reality (i.e., intensional information), and then describe the individual known facts of reality (i.e., extensional information). Finally, we list queries that an intelligent database system should be able to answer about this slice of reality.

3.1 STRUCTURE OF REALITY (A VIEW OF REALITY)

A scholarly work is uniquely identified by its title and authors, where the authors can be any number of scholars; each scholar is identified by a surname. No scholar can have two scholastic works with the same title.

There are different types of publications (media) in which works can be published, e.g., journals and monographs. Some works may be published in more than one publication, and some may not be published at all. Regardless of the medium, each publication must have at least two attributes, a title and a publication year.

A journal is identified by its title, volume number, and publication year. If two journals share the same title, then their volume numbers must be consistent with their publication years, i.e., a smaller volume number implies an earlier publication year, and vice versa.

A monograph is identified by its title, editors, and publication year. No two monographs can have the same title. In general, no two publications of different media can have the same title. For example, there cannot be a journal and a monograph with the same title.

A published work is identified by its work, the publication in which it appears, and the page numbers of the work within the publication. No two published works within the same publication can have overlapping page numbers. One published work can be referenced-by another. The year of the publication of the referenced work can be no greater than the year of the publication of the work referencing it. This relation (referenced-by) has a transitive closure, relevant-to, which can be directly derived from it.

3.2 FACTS OF THE REALITY

The extensional information to be represented in this simple database will be presented in tabular form. There are four known works, as shown in Table 1. Note that the labels to the left will be used to identify these works in other tables. There are two known journals and two known monographs as shown in Tables 2 and 3, respectively. There are five known published works, as shown in Table 4. Note that one work is published twice. There are three known references, as shown in Table 5.

2.6 B-EXPRESSIONS

Any expression is a B-expression if it is an atomic symbol (any alphanumeric word without blank) or a list of other B-expressions; i.e., B expressions are exactly the same as the lists in LISP. We denote the null list by < >, and a non-null list by $\langle a_1, a_2, \ldots a_n \rangle$, where the a_i can be any other list. We assume two operations on B-expressions:

(1) concatenation:

$$\langle a_1, a_2, \ldots, a_n \rangle * \langle b_1, b_2, \ldots, b_m \rangle \stackrel{\text{def}}{=} \langle a_1, a_2, \ldots a_n, b_1, b_2, \ldots, b_m \rangle$$

(2) component_of:

a in $\langle a_1, a_2, \ldots, a_n \rangle$
iff $a = a_i$ for some i.

2.7 NAMES

Any atomic symbol starting with an alphabetic character is a name. Some of the names can be used as variables, but there is no syntactic difference between variables and other names.

2.8 ADS OBJECTS

From the ADS user's point of view, the world consists of three categories of objects (what can be named and described by B-expressions):

(1) Assertions: a logical value either truth or falsehood, (denoted by T and F). The following logical operations are assumed to be available: $\wedge$(and), $\vee$ (or), => (implies), $\sim$(not), <=> (if and only if).

(2) Elements: any member of B-expressions. The following operations and relations are defined on elements: *(concatenation), in (component of), = (identity).

(3) Sets: any member of the power set of B-expressions, i.e., a set of B-expressions. The membership relation (ε) is the only relation defined on sets.

It is important to note here that (1) a set of sets is not an object in ADS, (2) no set operations are available in ADS, and (3) all B-expressions, including N, D and C, are elements; that is to say, the set of names (N), the set of descriptors (D) and the set of commands (C) are set objects of ADS.

2.9 DESCRIPTORS (D)

The user can *name* any object provided that the user can *describe* the object. A description of an object is called a descriptor (a-descriptors for assertions, e-descriptors for elements, and s-descriptors for sets). Each descriptor is encoded (by a fixed algorithm in ADS) and then represented by a B-expression (an element). In a similar way, LISP encodes a function specification (in the M-expression) and then represents it by an S-expression.

The syntax and the interpretation rule for the descriptors are fixed for ADS, but the semantics of a descriptor changes as the semantics of its components changes. The interpretation rule is called phi (Φ) and it corresponds to the function EVALQUOTE in LISP.

Any part of a descriptor which denotes an object under Φ is called an expression (a-expression, e-expression, or s-expression, for assertion, element, or set, respectively). As a set object, a-expressions are denoted by A_EXP.

The language for the descriptor is a first-order language in which variables range over the elements, not the sets or the assertions. A recursive definition of a name is allowed; i.e., the name associated with a descriptor may appear in the descriptor itself.

We call a list of variable declarations a form. For example, $\langle a{:}T_1, \langle b{:}T_2, c{:}T_3 \rangle \rangle$ is a form in which three variables are declared.

In the following definitions, we will use, for the sake of brevity, "x:T" as a representative of an arbitrary form, and p(x) will represent an arbitrary a-expression.

The syntax (in D) of descriptors is as follows.

(1) a-descriptors: ($\forall$ form) (a-expression)·
($\exists$ form) (a-expression)

Example:
($\forall$x:T)(p(x)): for all x in T, p(x) is true.
($\exists$x:T)(p(x)): for some x in T, p(x) is true.

(2) e-descriptors: (ι form)(a-expression)

Example:
(ι x:T)(p(x)): an x in T such that p(x) is true.

Note:
If for no element p(x) is true, the descriptor denotes nothing, and if there is more than one element satisfying p(x), any one (non-deterministically chosen one) can be denoted by the descriptor. In logic (e.g., Quine[16]), the expression (ι x) P(x) is called a 'definite description', and its meaning is 'the object x such that P(x) is true'.

(3) s-descriptors: (λ form) (a-expression)

Example:
(λx:T)(p(x)): the set of all x in T such that p(x) is true.

Note:
For the λ-notation, see Church.[17]

Note that all variables declared in the "form" have a scope which extends until the end of the descriptor in which the "form" appears.

At this point, Table 4 of the database is completely instanciated.

```
Referenced_by ==                                        (40)
    (λ<x: τ(Published-Work),
       y: τ(Published-Work)>)
    (x ≠ y ∧
     y.publication.year ≤ x.publication.year)
                                        →accept
Referenced_by ← <NDS_SP, ADT_AI>,                       (41)'
                <TT_AJM, NDS_SP>,
                <NSP_SP, NDS_SP>
                                        →accept
```

(40) defines the relation Referenced_by, and (41) makes specific references known. At this point, Table 5 of the database is completely instanciated. Note that although "Referenced_by" is defined to be a set name, it is used as a relation.

Up to this point, the scenario has been developing the structural and factual content of the database described in Section 3. The remainder of the scenario shows how derived information can be obtained.

```
Relevant_to ==                                          (42)
    (λ<pw1: τ(Published_Work),
       pw2: τ(Published_Work)>)
    (pw1 = pw2 ∨
    (∃pw: τ(Published_Work))
    (<pw1,pw> ε τ(Referenced_by)
     <pw,pw2> ε Relevant_to))
                                        →accept
```

(42) defines another relation on publications. Note the recursive nature of the descriptor. The relation Relevant_to is, in fact, the transitive closure of the relation Referenced_by.

```
? <TT_AJM, ADT_AI> ε Relevant_to                        (43)
                                        →Yes
? <NSP_SP, NDS_SP> ε Relevant_to                        (44)
                                        →Yes
? <NDS_SP, NSP_SP> ε Relevant_to                        (45)
                                        →No
```

In (43), TT_AJM is relevant to ADT_AI because TT_AJM is referenced by NDS_SP and NDS_SP is referenced by ADT_AI. In (44), NSP_SP is directly referenced by NDS_SP.

```
? (ι<form: FORM>) (∃any: ANY)                           (46)
        (μ(Journal) =
        <"λ",form,any>)
                    → <title: Phrase,
                       volume: Number,
                       year: Number>
```

In (46), we ask a question about the schema of Journal; what is its form? This is possible only because we know the encoding mechanism used in ADS. Given more detailed information about the encoding mechanism, other information can be extracted from a name's descriptor. For instance, the list of attributes (which is essentially what (46) extracts) can be obtained, or the list of all set names present in the a_expression can be obtained.

Given a database in this form, what kinds of things can be done? Obviously, new relations can be constructed from the information available. For instance, a relation stating which scholars reference other scholars can be defined (using "Referenced _by"). The structure of any database name can be extracted (as exemplified by (46)). Note that this can be done even if there are no specific instances of the particular kind of object currently in the database; this capability is available because the intension and extension are separated in ADS, and the intension is available (through the descriptor) once the name has been declared.

5. SUMMARY

5.1 CAPABILITIES OF ADS

ADS has a unique combination of features. Among these are:

- Database objects are assertions, elements and sets of elements. These three object types supply a flexible framework in which to state information, classify that information, and enforce constraints on the structure and organization of that information.
- ADS conceptually separates intensional and extensional information. It constrains extensional information to conform to intensional information. Intensional and extensional information are maintained in the same database.
- Descriptors are used to state intensional information. Recursive use of names is available within the descriptors. Descriptors are encoded and stored as elements. Thus, they can be manipulated and transformed like any other element.
- Quantifiers ('for all' and 'there exists') are available within descriptors to state complex relationships between database objects.
- Unstratified control is used in ADS. Commands, data, and data descriptors all are encoded in a common language. Although sets of sets are not primary ADS database objects, this construct can be 'simulated' by the use of unstratified control.

These features of ADS give it the following properties:

- Variant records and view translation are easily and naturally handled by ADS.
- Both local and global constraints are easily stated and enforced.

(start ≤ end) is not satisfied. This example illustrates how a record structure with imposed semantic conditions can be represented in ADS.

```
Scholar == (λx: Surname)                              (3)
                                          →accept
Scholar ← "Russell", "Dahl"                           (4)
                                          →accept
? τ(Scholar)                                          (5)
                                  →{Russell, Dahl}
```

The set name "Scholar" is defined as a name of the set of objects of the type Surname; two known members of Scholar are identified, and accepted by the system because their 'type' is correct and no condition is to be checked because the a_expression is 'null'; the known members of Scholar are then retrieved in (5).

```
Scholar_list ==                                       (6)
     (λ<head: τ(Scholar)> * (tail: ANY))
     (tail = < > ∨
      tail ε Scholar_list ∧ ∿(head in tail))
                                          →accept
? <"Russell", "Dahl"> ε Scholar_list                  (7)
                                          →Yes
? <"Dijkstra"> ε Scholar_list                         (8)
                                          →No
? <"Russell"> ε τ(Scholar_list)                       (9)
                                          →No
? <"Russell", "Russell"> ε Scholar_list              (10)
                                          →No
Scholar←"Dijkstra","Hoare","Tompa","Marsh"(11)
                                          →accept
?<"Dahl","Dijkstra","Hoare"> ε Scholar_list(12)
                                          →Yes
```

In (6), the set name "Scholar_list" is defined to contain lists of known members of Scholar such that no scholar appears more than once in the list. <"Russell", "Dahl"> matches the descriptor and is thus a possible member of Scholar_list (Note (5)). <"Dijkstra"> cannot be a member of Scholar_list because "Dijkstra" is not currently a known member of Scholar. <"Russell"> is not a known member of Scholar_list because there are currently no known members of Scholar_list, i.e., τ(Scholar_list) is the empty set. <"Russell", "Russell"> is not a possible member of Scholar_list because "Russell" appears twice. Note that (6) demonstrates the ADS capability of recursive type definitions, and dynamic record structure.

In declaration (13) below, the a_expression states that any new work cannot have an author that is also an author of a known member of Work with the same title. The declaration (13) demonstrates ADS's capability of specifying complex consistency conditions.

```
Work == (λ<title: Phrase,                             (13)
          authors: Scholar_list>)
        (∀ work: τ(Work))
        (∀ author: τ(Scholar))
        ((work.title=title∧author in authors)=>
             ∿(author in work.authors))
                                          →accept
```

```
TT==(ιx: Work)(x=<"Theory of Types",                  (14)
                  <"Russell">>)
                                          →accept
```

In (14), the element name "TT" is defined to be a name of a work and is intensionally 'bound' to a specific work which is an element. Since the descriptor of TT describes one and only one object, it can be thought of and used as a constant (similar to the CONST construct of Pascal). Similarly, the element names "NDS", "ADT" and "NSP" can be defined (see Table 1). Line (14) illustrates the usage of (ι x) notation.

```
Work←TT, NDS, ADT, NSP                                (15)
                                          →accept
```

In (15), four works are declared to be known members of Work. At this point, all of Table 1 is instanciated in the database.

In descriptor (16), the a_expression states that any new journal with the same title as any known member of Journal must have its volume number and year of publication ordered in an appropriate manner.

```
Journal==(λ<title: Phrase,                            (16)
           volume: Number,
           year: Number>)
         (∀x: τ(Journal))
         (x.title = title =>
          (x.volume<volume <=> x.year<year))
                                          →accept
AJM==(ιx: Journal)(x=                                 (17)
      <"American Journal of Mathematics",42,1936>)
                                          →accept
AI==(ιx: Journal)(x=                                  (18)
      <"Acta Informatica",13,1980>)
                                          →accept
Journal ← AJM                                         (19)
                                          →accept
Work ← AI                                             (20)
                                          →reject
```

Two journals (AJM and AI) are instanciated (as constants), and one (AJM) is declared to be a known member of Journal. In (20), AI is declared to be a known member of Work. However, the meaning of AI does not satisfy the descriptor for Work, and the command is rejected.

```
Monograph == (λ<title: Phrase                         (21)
                editors: Scholar_list,
                year: Number>)
             (∀mono: τ(Monograph))
             (mono.title ≠ title)
                                          →accept
SP == (ιx: Monograph)(x=                              (22)
          <"Structured Programming",
            <"Dahl","Dijkstra","Hoare">,1972>)
                                          →accept
LK == (ιx: Monograph)(x=                              (23)
          <"Logic and Knowledge",
            <"Marsh">, 1968>)
                                          →accept
Monograph ← SP                                        (24)
                                          →accept
```

	Title	Authors
TT:	"Theory of Types"	"Russell"
NDS:	"Notes on Data Structure"	"Hoare"
ADT:	"Abstract Data Type"	"Tompa"
NSP:	"Notes on Structured Programming"	"Dijkstra"

Table 1: Known Works

	Title	Volume	Pub.Date
AJM:	"American Journal of Math."	42	1936
AI:	"Acta Informatica"	13	1980

Table 2: Known Journals

	Title	Editors	Pub.Date
SP:	"Structured Programming"	"Dahl", "Dijkstra", "Hoare"	1972
LK:	"Logic and Knowledge"	"Marsh"	1968

Table 3: Known Monographs

	Work	Publication	Pages
TT-AJM:	TT	AJM	230, 265
NDS-SP:	NDS	SP	83, 174
ADT-AI:	ADT	AI	205, 224
NSP-SP:	NSP	SP	1, 82
TT-LK:	TT	LK	59, 102

Table 4: Known Published Works

Referent	Referrer
NDS-SP	ADT-AI
TT-AJM	NDS-SP
NSP-SP	NDS-SP

Table 5: Known References

3.3 QUERIES

What kinds of queries should an intelligent database system, in general, be able to answer about such a database? Below we simply list some queries we believe should be answerable by an intelligent system.

(1) Is Russell an author of Principia Mathematica?

(2) Is he the only author?

(3) Is there any work published in more than one publication?

(4) Does Dijkstra 'know' about any works published by Hoare?

(5) Which scholar publishes more than any other scholar?

(6) What are the currently known publication types?

(7) What are potential publication types?

(8) Is "technical report" a potential publication type?

(9) If I wanted to define a new publication type, say "book", what attributes would it have to have?

(10) What are the attributes of a Journal?

(11) Given a specific published work, what are all the published works that can be found through a chain of references starting at the original published work?

4. ADS REPRESENTATION OF THE PUBLICATION DATABASE

The following scenario is presented to show how the database described in the third section might be developed in ADS. We have 'borrowed' certain notations from Pascal, such as variable declarations and attribute selectors, to use as a part of D (the meta-language for the descriptors). Besides the system defined names, the following set names are assumed to be predefined:

Number: a positive integer

Phrase: any sequence of characters

Surname: the last name of a person

We will paraphrase $\tau(S)$, the extension of S, as "known S", and S itself, or the intension of $\hat{S}$, as "possible S". We prefix the response from ADS to the user by '→'.

```
Pages == (λ<start: Number, end: Number>)          (1)
         (start ≤ end)
                                          →accept
? <13,5> ε Pages                                  (2)
                                          →No
```

In (1), "Pages" is declared to be a set name whose elements are lists of two numbers; the first must be no greater than the second. (This will be used to specify the page numbers in "Publication"). In (2), <13,5> is compared with the descriptor of Pages to see if it is a possible member of the set. The answer is "No" because the a_expression

In (21), the a_expression states that no new monograph can have the same title as any known member of Monograph. Two monographs are instanciated (as constants), and one is declared to be a known member of Monograph. Note that in "Journal" and "Monograph" two attributes, "title: Phrase" and "year: Number", appear.

```
Medium ==                                              (25)
    (λtype_name: τ(SNAME))
    (μ(type_name)=<"λ",attrib:FORM,any:A_EXP>
     => ("title: Phrase" in attribΛ
         "year: Number" in attrib))
                                        →accept
Medium←"Journal", "Monograph"                          (26)
                                        →accept
```

In (25), the set name "Medium" is declared. A member of Medium can be any known set name (i.e., τ(SNAME)) such that its descriptor requires any element associated with it to have a title and a publication year. "Journal" and "Monograph" are two such set names, and in (26) they are declared to be known members of Medium. Conceptually, Medium is a realization of the 'classification' concept of Smith.[6] Medium will be used as a generic 'access function' to obtain the specific publication associated with the known members of Medium. For instance, it would be possible to define a new set name "BOOK" whose 'form' contains a 'title' and a 'year'. Then "BOOK" could be declared to be a known member of Medium.

This capability of ADS to treat a 'meta-expression' (an expression to specify a set of expressions, for example an s-descriptor) as an element expression, is one of the most important and unique ones compared with other data models. Thus, (25) illustrates ADS's capability of unstratified control.

```
Publications = (λpub: ANY)                             (27)
               (∃type: τ(Medium))
               (pub ε τ(type))
                                        →accept
Publication_Name == (λname: Phrase)                    (28)
               (∃pub: Publication)
               (name = pub.title)
                                        →accept
```

In (27), we state that a member of Publication can be anything that is a known member of any known member of Medium (see (29) and (30) below). In Smith and Smith's terminology[5], Publication is a generalization of Journal and Monograph (i.e., the result of set union operation). In (28), the set name "Publication_Name" allows the user to 'select' or 'strip off' the title of members of Publication.

```
? AJM ε Publication                                    (29)
                                        →Yes
? AI ε Publication                                     (30)
                                        →No
? Publication_Name                                     (31)
            →{"American Journal of
            Math.", "Structured Pro-
            gramming"}
```

Note that AJM is a member of Publication since it is a known member of Journal. AI is not a member of Publication since it is not a known member of Journal or Monograph.

```
Journal ← AI                                           (32)
                                        →accept
Monograph ← LK                                         (33)
                                        →accept
? AI ε Publication                                     (34)
                                        →Yes
```

As we add AI to be a known member of Journal (in (32)), it becomes a member of Publication. At this point, Tables 2 and 3 of the database are completely instanciated.

In (35), an assertion is made that states that no two known members of Publication derived from different known members of Medium can share the same title. For instance, no member of Journal can have the same title as any member of Monograph. In (36), the assertion is declared to be true.

```
Disjoint_Publication_titles==                          (35)
     (∀x: τ(Medium))
     (∀y: τ(Medium))
     (x ≠ y => (∀u: τ(x)) (∀v: τ(y))
               (u.title ≠ v.title))
                                        →accept
Disjoint_Publication_titles:=True                      (36)
                                        →accept
```

Judgement (36) would be rejected if there already are known members of Journal and Monograph with the same title. Since (36) is accepted, the assertion becomes a constraint and will be tested after each transaction.

In descriptor (37), the a_expression states that no two known members of Published_Work can have the same publication and overlapping pages.

```
Published_Work==                                       (37)
     (λ<work: τ(Work),
        publication: Publication,
        pages: Pages>)
     (∀pw: τ(Published_Work))
     (publication ≠ pw.publication ∨
      pages.end < pw.pages.start∨
      pw.pages.end < pages.start)
                                        →accept
TT_AJM==(ι x: Published_Work)                          (38)
        (x=<TT,AJM,<230,265>>)
                                        →accept
```

In (38), the element name "TT_AJM" is defined to be the name of a member of Published_Work whose work is TT, publication is AJM, and pages are 230 to 265. Similarly, the element names "NDS_SP", "ADT_AI", "NSP_SP", and "TT_LK" can be defined (see Table 4).

```
Published_Work ← TT_AJM, NDS_SP, ADT_AI,               (39)
     NSP_SP, TT_LK
                                        →accept
```

a specification of a function or as an argument depending on where the S-expression occurs.

This capability of 'unstratified control' is one of the most important and unique properties of ADS. Due to this capability, ADS can incorporate in the database not only a schema of the data, but also the schema of the schema, and so on. Also, it can provide a capability of multi-level abstraction of reality.

2.3 INTENSION AND EXTENSION

There are two kinds of judgements users make in general about reality: intensional judgements and extensional ones. Roughly speaking, intensional judgements represents 'general facts' such as "every journal has a publication date", "no two books have the same title" and "a brother of the father is an uncle." On the other hand, extensional judgements represent 'specific facts' such as "CACM Vol. 22 was published in 1979," "Madnic and Donovan wrote the book Operating Systems, Tsichritis and Bernstein wrote the book Operating Systems", and "Joe is a brother of my father." Another way of differentiating the two kinds of judgements is that the former represents a user's view of how the world is supposed to be, and the latter represents a user's view of how the world actually is. The database manager, a part of ADS, checks the consistency between extensional judgements and related intensional judgements automatically i.e., without user specification. For example, if the intensional judgement "all pants are made of cotton" is entered into the database, then, subsequently, the extensional judgement "my pants are made of orlon" will be rejected by the manager because the extensional judgement is not consistent with the previous intensional one. At this point, the user has two options. One is to cancel the intensional judgement and enter a new one. The other is to change the extensional judgement.

2.4 ADS AS A STATE MACHINE

ADS models an information system at a low conceptual level, and it is not intended that any human user will interface with ADS directly. Thus, the relationship between the information system and ADS might be depicted as in Figure 2.

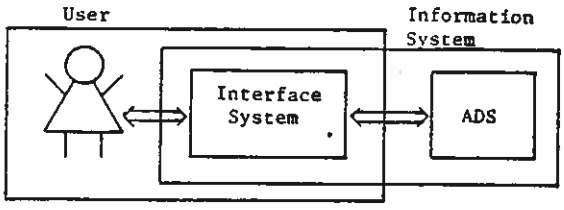

Figure 2: User and ADS

In the remainder of this paper, the term 'user', when applied to ADS, refers to the interface system, and not a human user.

The specific model of ADS described in this paper is a single user model, and can be thought of as a state machine (Figure 1).

A single sequence of commands is input to ADS, and a single sequence of responses is output from ADS. Commands may be either update commands or query commands.

As a state machine, ADS has an interpretation function ϕ, which has two inputs, the command and the current information state, and two outputs, the response to the user and the new information state. If the command is a query command, then the appropriate information is extracted and becomes the response to the user; the new information state is the old (current) information state. If the command is an update command, then a potential information state is created and checked for consistency. If the potential information state is consistent, then it becomes the new information state and the response 'accept' is produced to the user; otherwise, the new information state is the old (current) information state and the response to the user is 'reject'. The information state is what is commonly thought of as the database itself.

2.5 BASE LANGUAGE

In ADS the base language is used primarily to represent a user judgement, which involves objects, names of objects, descriptions of objects and the relationships among the objects (e.g., a certain symbol object is a member of a certain symbol class).

Some expressions in B are also used to interface with the database manager prescribing what operation the manager is requested to perform on the database.

Thus, B is partitioned into several blocks based on its usage by the users and the database manager; Names ($N \subset B$) for naming objects, Descriptors ($D \subset B$) for describing objects, and Commands ($C \subset B$) for prescribing actions for the manager.

In the formal definition of ADS[1], we adopt the set of binary trees (any symbolic representation thereof) as the base language due to its structural simplicity. In this paper, however, we will introduce different meta-languages for B, N, D, C, denoted by $\underline{B}$, $\underline{N}$, $\underline{D}$, $\underline{C}$, for ease of understanding, and we assume that there exists an encoding function from these meta-languages to the binary trees. Also, for the sake of brevity, we assume that the encoding of $\underline{D}$ into the binary tree is defined in two steps; i.e., $\underline{D}$ is encoded into $\underline{B}$ first, then $\underline{B}$ is encoded into B. Hereafter, we deal only with the meta-languages $\underline{B}$, $\underline{N}$, $\underline{C}$ and $\underline{D}$ and the encoding of $\underline{D}$ into $\underline{B}$.

The encoding of an s-descriptor into a B-expression is (here, [ ] denotes the encoding function):

$$[(\lambda\ \text{form})(\text{a-expression})] \triangleq$$
$$\langle\text{"}\lambda\text{"}, [\text{form}], [\text{a-expression}]\rangle$$

Among others, the following special objects are given system names:

(1) ANY: the set of all B-expressions

(2) SNAME: the set of set names

(3) FORM: the set of forms (variable declaration sequences; encoded in B)

(4) A_EXP: the set of a-expressions (encoded in B)

2.10 DECLARATIONS (JUDGEMENTS)

The user interacts with ADS through either a transaction or a query. A transaction is a sequence of declarations (representations of judgements).

The user enters an intensional judgement into ADS whenever a new specific abstraction is made about the world, i.e., whenever the user recognizes (in conception) the significance of a particular way of viewing a particular object. The user represents that view by constructing a descriptor for the object, and further, assigning a name to the object. Intensional judgements in ADS are represented by 'definitions', and the syntax of a definition is:

name==descriptor

which is in an encoded B-expression form when ADS accepts the judgement. ADS rejects the above definition (intensional judgement) when the name is already defined.

The effect of entering an intensional judgement into ADS is the establishment of an association between the name and the descriptor. The association is called mu(μ). After the definition, the following holds:

$$\mu(\text{name}) = \text{descriptor},$$

and the name becomes an a-name if the descriptor is an a-descriptor, and similarly for e-name and s-name.

The object denoted by the descriptor, i.e., Φ(descriptor), is called in ADS the intension of the name, and we denote it by $\Phi(\mu(\text{name}))$. $\Phi(\mu(\text{name}))$ denotes what 'possible' objects can be associated with the name.

The user enters an extensional judgement into ADS whenever a specific observation is made about the world.

Associated with each name, there exists an object called the extension of the name, which represents the accumulation of the extensional judgements so far entered into the system. We denote the association between a name and its extension by tau(τ), and let τ(name) represent the extension of the name. It denotes what 'known' object is currently associated with the name.

Initially, when the name is defined, the following hold:

τ(a-name) = undefined,

τ(e-name) = undefined,

τ(s-name) = the empty set.

Every extensional judgement involves one name whose extension is changed if the judgement is accepted by ADS. The syntax and semantics of an extensional judgement are as follows:

(1) a-name:=T(or F)

If $\Phi(\mu(\text{a-name})) = T$ (or F) and τ(a-name) = undefined then assign T (or F) to τ(a-name).

(2) e-name:=e-expression

If $\Phi(\mu(\text{e-name})) = \Phi(\text{e-expression})$ and τ(e-name) = undefined then assign Φ(e-expression) to τ(e-name).

(3) s-name ← e-expression

If $\Phi(\text{e-expression}) \in \Phi(\mu(\text{s-name}))$ then include Φ(e-expression) as a member of τ(s-name).

The user enters a query to ADS for retrieving an object from the database. Syntactically, a query consists of a question mark followed by a single expression.

The effect of the query is a display of the denotation of the expression to the user.

2.11 CONSISTENCY AND CONSTRAINTS

The conditional part of each operation in the above definitions is a part of the consistency condition every extensional judgement must satisfy in order that ADS may accept the judgement. An individual extensional judgement will be accepted by ADS if and only if its syntax is correct, every name in the judgement is defined, every expression has a denotation, and the following consistency condition is satisfied.

Consistency Condition:

For each user defined name,

$\tau(\text{a-name}) = \Phi(\mu(\text{a-name}))$ or undefined,

$\tau(\text{e-name}) = \Phi(\mu(\text{e-name}))$ or undefined,

$\tau(\text{s-name}) \subseteq \Phi(\mu(\text{s-name}))$.

ADS has very flexible data object types. Variant records and view translations are easily and naturally handled by ADS. Recursive definitions of database objects are possible, and quantifiers (for all and there exists) are available for stating complex relationships among objects. Through these features, ADS supports the three abstraction mechanisms proposed by Smith & Smith for a general purpose database system: aggregation, generalization and classification.[4,5,6]

ADS uses the same language for stating intensional (conceptual or schematic) and extensional (factual or specific) information. Thus, it is possible for the user to be his/her own database manager.

The property of unstratified control[7] allows the user to define any level of abstraction desired; for example, by introducing a schema for schemata. Researchers may frequently change their view of the structure of reality as they sift through the pieces of a puzzle that will fit together only when apprehended in the correct way. Such researchers must have the capability to restructure the way in which the database interprets the information being manipulated.

The ability to perform semantic checking on information is undeniably important. Even in relatively static databases, such as admission records of patients, editing of input data is required. Humans are inherently error prone, and they need help to confirm their statements of fact. A user should not be allowed to enter the value 155 (i.e., the patient's weight) for the patient's temperature. The problem is even more acute in a dynamic situation where the user may be thinking of a previously defined conceptual framework. Without the guideposts of semantic constraints in a complex, ever-changing structure, the researcher will inevitably stumble or get lost.

ADS supplies two mechanisms for applying semantic constraints. The first is the manner in which intensional and extensional information is coordinated. Not only does ADS separate intensional information from extensional information, but it requires all extensional information to conform to the intensional information. The second is that ADS allows the user to state global assertions that constrain relationships among different database objects.

The medical field is not a static one. The perception of a disease, its cause, treatment, and cure changes dynamically with time. Any database system that claims to address these concepts must be able to adapt to changes and refinements. ADS, we believe, will supply a framework capable of meeting these needs.

1.2 OTHER SEMANTIC DATA MODELS

ADS has the following primary capabilities.

(1) Abstraction mechanism: aggregation, generalization, and classification,

(2) Extensive semantic consistency checking,

(3) Explicit separation of the concepts of intension and extension, combined with the integration of the intensional and extensional databases, and

(4) Unstratified control over language interpretation.

There are other known models that possess some of the above capabilities, but none have all of the capabilities, as far as we know. Smith & Smith[8] incorporate abstraction mechanisms and semantic consistency checking which is similar to type checking in a programming lanugage. However, the notions of intensional and extensional databases are not dealt with directly, not to mention the possible integration of the two within a single database.

A similar observation can be made on the extended relational model of Codd.[9] It has more semantic consistency checking capabilities and more abstraction capabilities (although only up to 3 levels) than the original relational model[10], but again, no effort is made to separate and integrate intensional and extensional information at the user's level.

The integration of intension and extension has been more successfully done in logical database models such as Minker's.[11] However, the abstraction mechanisms are not its main concern, and the unstratified control capability is not present.

There are two known data models which have the capability of unstratified control; Abrial[12] and Laine.[13] In particular, Abrial's is an extensive, powerful and valuable data model worth careful study, not only because of its unstratified control but also because of its inclusion of a 'process' concept as a part of a database model. Nevertheless, it fails to recognize the importance of the intension/extension dichotomy in the database design.

Some recently proposed data models, such as those of Roussopoulos[14] and Mylopoulos[15], treat the notions of intension, extension, and abstraction as primary concepts in the database design; but again they lack the generality of uniform treatment of the meta and object databases that can be facilitated by unstratified control.

1.3 CONTENT OF THE PAPER

Section 2 presents an overview of ADS and states what types of database objects are available, how they are defined, and how they are manipulated. Section 3 presents a narrative of a sample database to be modeled. Section 4 presents a scenario of how this sample database might be modeled in