1989

The development of a knowledge-based system for semantic data modeling /

William H. Fenton
Lehigh University

The Development of a Knowledge-Based System for Semantic Data Modeling

By

William H. Fenton

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University

December 1988

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

January 8, 1968
(date)

John L. Clayton
Professor in Charge

M C Thomas
Chairman of Department

Table of Contents

Abstract

The basis for a sound and logical organization of a corporation's data is the building of logical data structures. Logical data structures support the data definitions and data relationships as related to an organization's business processes and functions.

Organizations implement business processes and functions using database management systems (DBMS). DBMS's are designed and engineered to manipulate data from a physical aspect and do not represent the meaning of the data. The semantics of business processes and data definitions are embedded in program code and replicated across systems. This creates inconsistency and unnecessary expenditures in data processing resources.

Limited research has been devoted to the development of a Semantic Data Model (SDM) to support business processes and data definitions. Information about the business and data is left to individual interpretation by systems analysts.

This thesis deals with the development of a knowledge-based system to assist in recording the definitions, relationships, and semantics of data in an organization. The benefits of such a system are: (a) user involvement in data definition; (b) the reduction of ambiguous data; and (c) the basis for the implementation of a physical database, regardless of the DBMS.

Chapter I. Semantic Data Structure Background

Today, many organizations and companies develop information systems to establish a position within their market place or to provide a service that is new. Either way, the organization has an opportunity to gain an edge over its competition. In doing so, information systems departments are receiving requests to analyze and develop systems or change the current data processing services their respective departments receive.

The operating department requests focus on the individual needs of each department. In each case, the request satisfies a department requirement, contributing to achieving its business goal. Typically, how a request is satisfied and at what cost are overlooked.

When analysts review all department requests, they discover there is duplication of effort and redundancy of information within the organization's information systems. With duplication of effort, inconsistencies arise between data and the rules to process and analyze that data. These information systems interface and exchange data only when required, and usually only after a political struggle occurs over data ownership.

Following is an actual case of how a company attempted to develop a corporate information system.

Company X developed a new customer database. The system is intended to support and monitor the development of, customer contracts from a customer's initial request through completion. The customer system contains a price module. The price module records the price for computer hardware repair services. The system was developed and implemented under the assumption that prices will not change for each service charged, and that eighty percent of the contracts will use a standard price.

The new customer database system was implemented but never used because of two design oversights. The first was due to an original assumption. The company found that in practice only twenty percent of the contracts were written with standard pricing, while the remaining eighty percent were variations of the standard price. Secondly, the price module was only one of ten modules to be developed. When work began on the other modules, it was discovered that the price data did not meet the requirements to support implementation of the system. A project reassessment estimated it will cost the company $1.5 million over their budget to redesign the system.

The problem company X faces is that the time and effort needed to develop a corporate data model consistent with the business rules (i.e., the way business is actually conducted), is reflected in the projected expenditure of $1.5 millon needed to revise the system. What may have been a small cost to change in the early part of the system's life cycle will now cost three to five times as much. Information is

segregated among various files; hence, the high cost in time and dollars to evaluate the impact of the scope change. Note that this is the fifth time in six years the system has been designed and implemented.

Many systems fail because the company organizations do not expend the necessary time and effort to design and develop a corporate data model. Without a data model, departments cannot take advantage of the corporation's data. With continued problems of inconsistent and mistimed information, corporations will continue expending millions of dollars to develop independent information systems, instead of integrated information systems.

Chapter II. Problem Statement

A data model defines the entities, attributes, and data relationships of a business. The data meaning, usage, and relationships are left to individual interpretation. This opening allows multiple meanings of the data to exist. As a result, analysts can design and implement independent applications; but, in an effort to reduce costs, the meaning of the data must be extracted and documented so that it will be given the same interpretation in each application.

For example, consider a car sale. To a salesman, the sale of the car helps to meet a sales quota and to obtain a commission. To the bank, the car is collateral for a loan. To a mechanic, it is future work in the form of repairs. In each case the car is viewed in a slightly different manner.

> In recognition of the critical need for a semantic database model, a number of researchers have recently considered the design of such a model [Abrial 1974, Bachman 1977, Biller+Neuhold 1978, Chen 1976, Codd 1979, El-Masri+Wiederhold 1979, Hammer+McLeod...] Although there are substantial and important differences in the detail of these various semantic database models, they have all attempted to increase the degree to which a schema captures the meaning of data in a form intelligible to its designers and users [18, pp. 194].

To assist in the derivation and to capture the meaning of business data, Hammer & McLeod have developed a formalism called 'Semantic Data Model' (SDM) [13]. They declare that the semantics of an organization's business data can be represented by a data model's

entities, attributes, and data relationships; and SDM will provide more of the meaning of business data than other data models of the current day.

Hammer and McLeod declare relationships between data extend farther than data model entities. In contrast, Appleton defines a relationship between entities as Business Rules [2]. The Business Rules explain the constraints which exist among zero, one, or more entities. The Business Rule controls the information contained within the business model, and what data will be created from the business model. Similarities exist between SDM and Business Rule methods, as shown in Figure 1.

Hammer/McLeod	Appleton
Class	Entity
Attribute(s)	Attribute(s)
Interclass Connections	Business Rules

Figure 1

Each defines an entity or class (i.e., Purchase Order, Customer, Part Number). Attributes describe the characteristics of the entity or class (i.e., Name, Address, City, State). In addition, SDM extends attributes to the entire entity called Class Attributes and occurrences of each entity called Member Attributes. There is a difference between Interclass Connections and Business Rules.

Hammer and McLeod extend SDM relationships and constraints to the entity and attribute level. They emphasize that attributes belong to a class which is bound by constraints. These constraints exist not just between entities but, as stated by Appleton attributes as well. This SDM recursive format documents the data interrelations throughout the data model, thereby extending the business rules to a class and an attribute.

Chapter III. SDM Rule Implementation

Hammer and McLeod developed a detailed methodology. To understand and document the methodology requires reviewing the entire publication and the SDM syntax. The SDM syntax alone will not provide the entire set of constraints for a class or attribute to exist. The major task in the development of the SDM/Prolog system is to translate the SDM syntax (Appendix A) and validation criteria into a set of Programming Logic (Prolog) rules.

Prolog is an interpretive programming language developed at the University of Marseille by Alan Colmerauer in the early 1970s [8,9]. Since 1970 a group at the University of Edinburgh has made several improvements to Prolog, which is considered to be the defacto standard [16].

Prolog is a declarative language, where the programmer specifies what the program is supposed to achieve by using a number of facts and rules to find a solution to the problem. Compared to procedural languages -- such as Pascal, Basic, or Cobol -- Prolog instructs the computer what to do, not how to do it.

The intent of Prolog, used as a knowledge-based system for business systems, is to capture as much knowledge of business rules as possible from an expert, and translate that knowledge into rules and facts the

computer understands in order to assist or simulate the decision-making process.

The particular version of Prolog used for this system is Turbo Prolog [5,6] for personal computers. Turbo Prolog departs from the Edinburgh standard, particulary because it requires strong data typing. This feature is required in order to compile code.

To manage SDM/Prolog development, each SDM rule is divided into small manageable sections and placed into a hierarchical structure according to function (see Figures 2 and 3 at end of the chapter). To translate the rule hierarchy into Prolog syntax requires that an analyst first translate the rule into an English sentence. Prolog lends itself more readily to implementation of a rule if that rule is first translated into an English-like structure.

Rule translation uses a numbering system similar to the standard paragraph numbering system. When a new subsection is created, a letter or number is appended to the rule. For example, Rule_1A is the second rule within Rule_1. This technique groups rules within the same category and provides a cross reference between the rule's English translation and the Prolog implementation of the rule. When a rule is used more than once, it is generalized and documented by a comment. Each rule implemented in the SDM/Prolog system is translated into in English-like sentence structure (see Appendix C). For example, the

English translation of a valid SDM class name whether it is a Base Class or Nonbase Class is:

> **Each CLASS has a CLASS NAME. A CLASS NAME is composed of uppercase letters and underscores. A CLASS NAME must be unique with respect to all CLASS NAMEs used in the business model. We will call this Rule_1.**

The implementation of Rule_1 in Prolog will be a series of predicates. For the rule to be true in Prolog, all of the predicates which comprise it must be true. One possible translation is:

```
rule_1:-
     lineinput(20,20,45,7,7,"Class name: ","",INPUT),
     upper_lower(INPUT,CLASS),
     isname(CLASS),
     class_list(LIST),
     not(member(CLASS,LIST)),
     retract(class_list(LIST)),
     assertz(class_list([CLASS|LIST])),
     write("Class added"),
     !.

rule_1:-
     write("Class name is invalid or already exists").
```

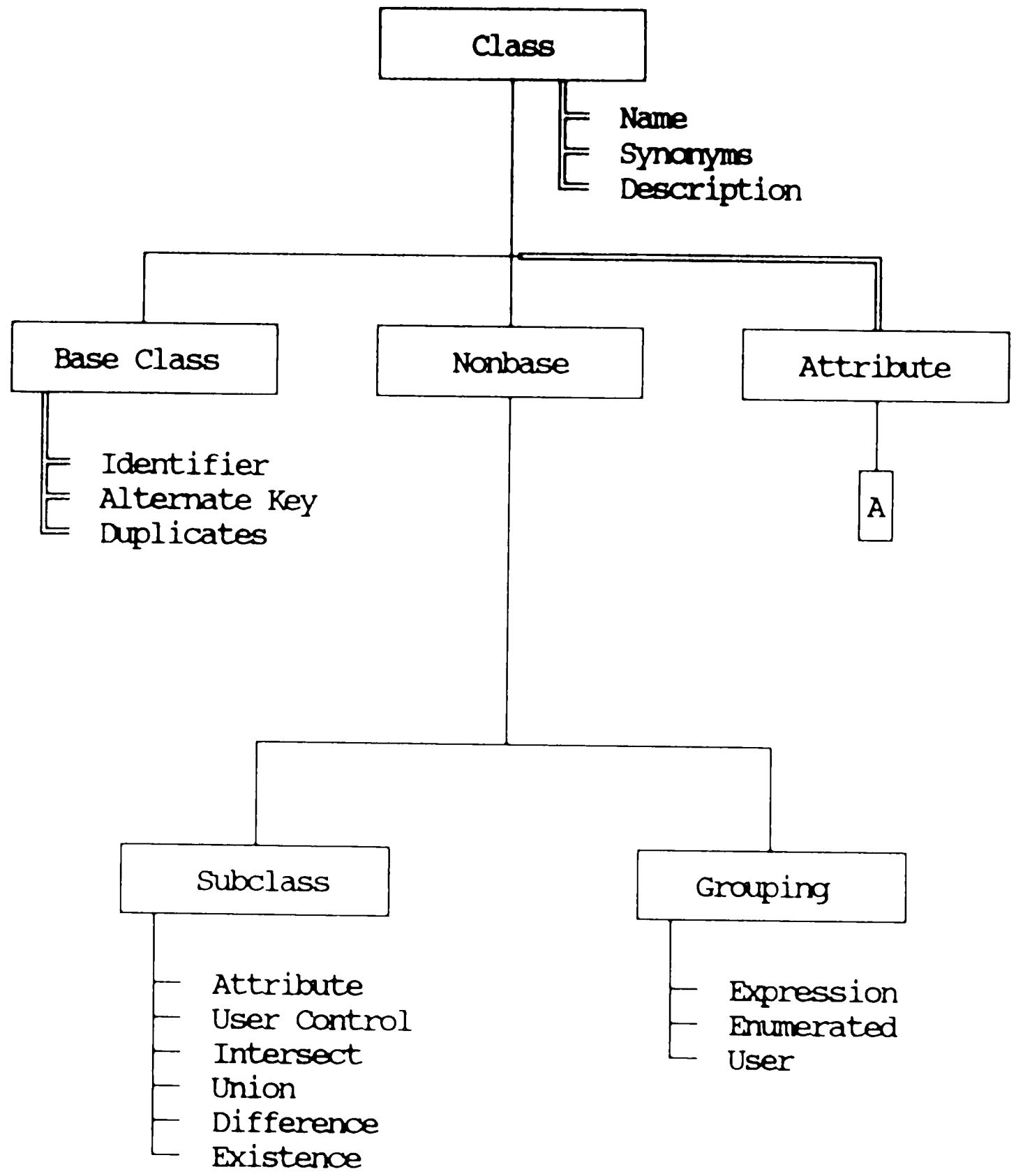
Lineinput is a screen input predicate documented in Turbo Prolog Toolbox [6, pp. 256], Class_list is a Prolog database predicate, and Member is predicate to check whether X is a member of a Prolog list,

discussed in Clocksin and Mellish [10, pp. 49]. All other predicates are documented in Turbo Prolog [5].

The Prolog rule can be read as follows:

> "Prompt the user for the class name, convert the class name to lower case for validating and processing, check that the class name is composed only of letters and underscores, retrieve the existing list of classes from the database, insure the class name is unique, delete the old class list, create the new class list by concatenating the new class and old class list, write a message, and force Prolog to stop backtracking.
>
> If the first occurrence of the rule_1 predicate fails, write an error message."

SDM Features Hierarchy Diagram



Legend: Functions listed by a double line (═) are common SDM features. A single line (—) indicates one of the following must be selected, e.g. Class has the common features Name, Synonyms, and Description and is either a Base Class or Nonbase Class).

Figure 2

SDM Features Hierarchy Diagram (cont.)

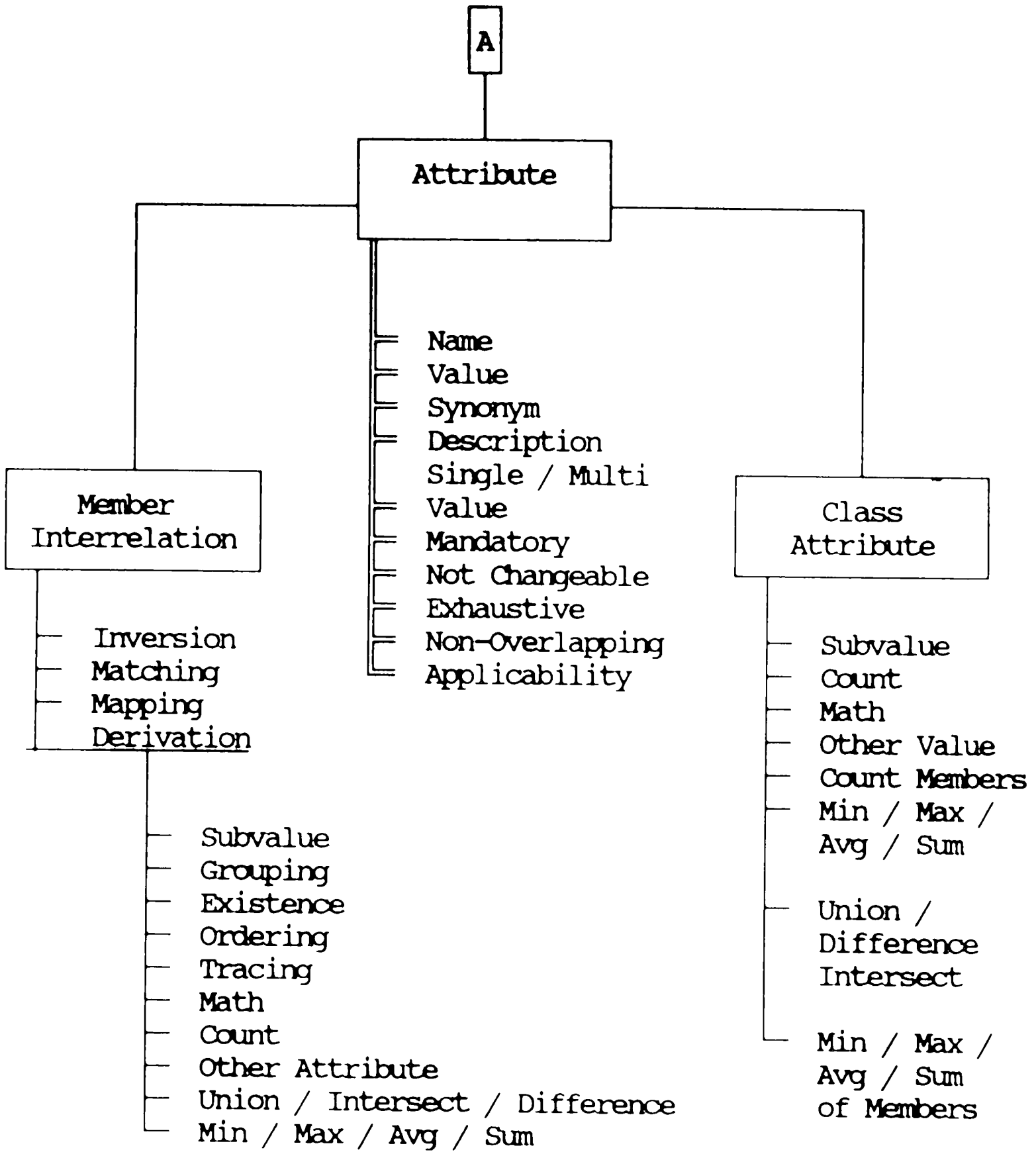


Figure 3

Chapter IV. System Overview

To test the SDM/Prolog system, this thesis will use the Tanker Application example of Hammer and McLeod [13]. This application is designed to monitor and control ships that enter or leave the United States coastal waters. A copy of the Tanker System example can be found in Appendix B.

To validate the Tanker Application, SDM/Prolog is designed to be an interactive system using question and answer menus to define an SDM class or attribute. A selection is processed by placing the cursor at the function or data value. This menu selection method provides a simple way to reduce rule coding, validate data, and control flow. If a selection is aborted, the system ignores any changes and returns to the main system menu. SDM/Prolog supports on-line reporting. It allows a user to list data from the database as the system is built. The limited reporting capabilities route the information to the screen.

Chapter V. Observations

An alternate validation technique would be to design and implement the system by translating the SDM Tanker Application into Prolog clauses and have the SDM/Prolog rules validate the clauses. This technique was tested, but proved to be cumbersome and produced complicated rule definition.

SDM rules are flexible, with complexity based on limited data requirements. A simple form of a Base Class requires a name, member attribute, and logical key. Defining a Nonbase Class requires a name and interclass connection. A more complicated format is a member attribute, with an attribute predicate defined with a mapping clause. The logic rules coded into the SDM/Prolog system were input and/or output driven, rather than deriving knowledge from the knowledge expert to build a knowledge base similar in functionality to the Berkshire MECHANIC automobile knowledge-base system [3].

The degree of interpretation of a class, attribute, or data relation still remains with an individual's business knowledge, experience, and expertise. In the SDM Tanker Application, the Nonbase Class HULL_NUMBERS is defined as the:

> interclass connection: subclass of STRINGS where format is number where integer.

An alternative would be:

> interclass connection: subclass of INTEGER.

SDM/Prolog's goal is to assist in managing these various interpretations and to document the meaning of data so that it will be given the same interpretation in every application.

Chapter VI. Future Enhancements

This thesis lays the foundation work for future enhancements and provides further functionality not addressed here. Due to time constraints, the scope of the system was defined that not all functions of SDM are included. Areas of SDM enhancements and directions for further research include:

o Help System

Without a detailed background of SDM and knowledge of Prolog, the user is limited to understanding the system functions. A Help system that encompasses the overall system, each functional area, and field level help would enhance the system. This function could be developed using an in-house help system, or the help system documented in the Turbo Prolog Toolbox [6].

o Screen Design

Design the system to display a full screen of data instead of responding to one question at a time, as shown in Figure 4. This will reduce the number of user responses and speed data entry. The suggested method is to use the screen definition and manipulation facilities documented in Turbo Prolog Toolbox [6].

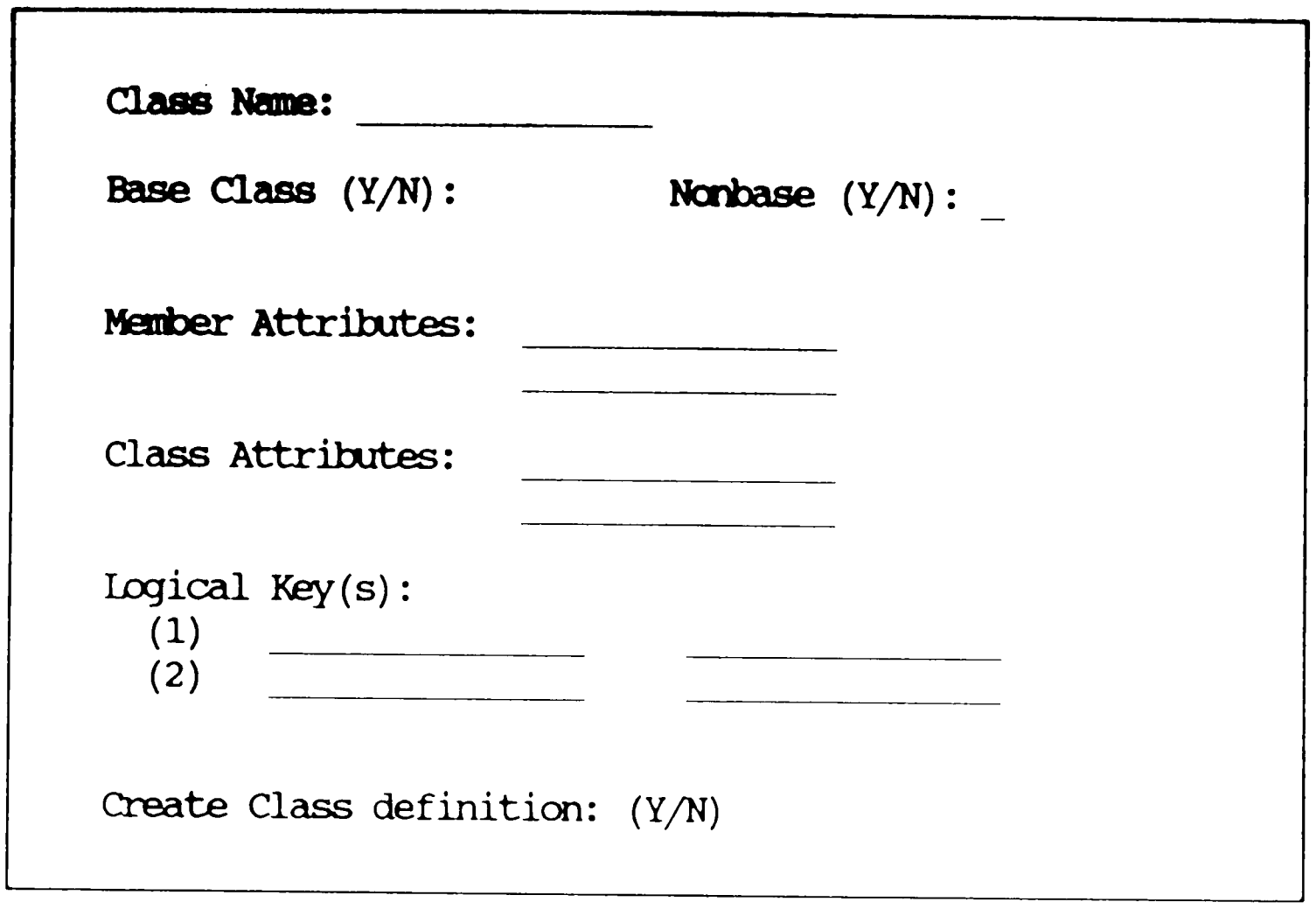


Figure 4

o Reports

Reports are generated on-line only. Additional reports may be on-line and printed:

Cross Reference
Value class to member attribute
Member attributes to Class
Base Class
Nonbase Class

o System

Isolate the input/output activity from the knowledge base for queries. One Prolog system may be written to manage the creation of the knowledge base and another to manage the queries.

Chapter VII. Conclusion

The goal of this thesis was to demonstrate that Hammer and McLeod's Semantic Data Modeling methodology, based on a structured syntax and set of constraints, can be implemented using Prolog, in a system called SDM/Prolog.

The results of documenting the SDM Tanker Monitoring Application as a skeletal semantic database model, using the SDM/Prolog system, supports business data and rules can be extracted from a business knowledge expert and recorded in a knowledge based system such that they will be given the same interpretation in each business application.

Chapter VIII. Location of SDM/Prolog System

In compliance with thesis preparation guidelines, the following materials are on file in the office of thesis advisor Professor John C. Wiginton.

a. two SDM/PROLOG system object code diskettes,
b. two SDM/PROLOG source code diskettes,
c. SDM/PROLOG program source listings,
d. Thesis

Professor John C. Wiginton
Room 479
Department of Industrial Engineering
Mohler Lab #200
Lehigh University
Bethlehem, Pennsylvania 18015

Chapter IX. References

[1] Appleton, Daniel E., "Business Rules: The Missing Link," Datamation, October 15, 1988, pp. 145-150.

[2] ------, "Rule-Based Data Resource Management," Datamation, May 1, 1986, pp. 86-99.

[3] Berkshire, Turbo Shell, (Berkshire Software, Lynbrook, New York, 1987).

[4] Berman, Sonia, "A Semantic Data Model as the Basis for an Automated Database Design Tool," Information Systems, Vol. 11, No. 2, 1986, pp. 149-165.

[5] Borland, Turbo Prolog 2.0, IBM Version (Borland International, 1988).

[6] ------, Turbo Prolog Toolbox, IBM Version (Borland International, 1987).

[7] Brodie, Michael L., "The Application of Data Types to Database Semantic Integrity," Information Systems, Vol. 5, No. 4, 1980, pp. 287-296.

[8] Colmerauer, A., "Les Systemes-Q ou un Formalisme pour Analyser et Synthesizer des Phrases sur Ordinateau," Publication Interne No. 43, Dept. d'Informatique, Universite de Montreal, Canada, 1973.

[9] Colmerauer, A., Kanoui, H., Pasero, R., and Roussel, P., "Un Systeme de Communication Homme-machine en Francaie," Research Report, Groupe Intelligence Artificielle, Universite Aix Marseille II, France, 1973.

[10] Clocksin, William F. and Mellish, Christopher S., Programming in Prolog, third revised and extended edition (Springer-Verlag, 1987).

[11] Doe, Lawrence W., Glemser, Raymond G., and Wiginton, John C., "A Knowledge-Based System for Evaluation of the Plan of Internal Control," working paper 87-001, Lehigh University, Bethlehem, Pennsylvania, June 15, 1987.

[12] Frost, R. A., "Using Semantic Concepts to Characterize Various Knowledge Representation Formalisms: A Method of Facilitating the Interface of Knowledge Base System Components," The Computer Journal, Vol. 28, No. 2, 1985, pp. 112-116.

[13] Hammer, Michael, and McLeod, Dennis, "Database Description with SDM: A Semantic Database Model," ACM Transactions on Database Systems, Vol. 6, No. 3, September 1981, pp. 351-386.

[14] Kowalski, Robert, "AI and Software Engineering," Datamation, November 1, 1984, pp. 92-102.

[15] Kroenke, David, Database Processing: Fundamentals, Design, Implementation, second edition (SRA, 1983).

[16] ------, Logic for Problem solving (North-Holland, 1979).

[17] McLeod, Dennis, "High Level Definition of Abstract Domains in a Relational Database System," Journal of Computer Languages, Vol 2., No. 3, 1977).

[18] ------, and King, Roger, "Applying a Semantic Database Model," Proceedings of the International Conference Entity-Relationship Approach to Systems Analysis and Design, Los Angeles, California, December 1979, pp. 193-210.

[19] Shafer, Dan, Turbo Prolog Primer, revised edition (Howard W. Sams & Co, 1987).

[20] Schur, Stephen, "Intelligent Databases," Database Programming & Design, June 1988, pp. 46-53.

[21] Ten Dyke, Richard P., "Outlook on Artificial Intelligence," Journal of Information Systems Management, Fall 1984, pp. 10-16.

[22] Yin, Khin Maung, with Solomon, David, Using Turbo Prolog (Que, 1987).

[23] Weiskamp, Keith and Hengl, Terry, Artificial Intelligence Programming with Turbo Prolog (John Wiley and Sons, 1988).

APPENDIX A SDM Data Definition Language Syntax

(Source: Hammer and McLeod, [12, pp. 381-384])

Legend

(1) The left side of a production is separated from the right by a "<==".

(2) The first level of indentation in the syntax description is used to help separate the left and right sides of a production; all other indentation is in the SDM data definition language.

(3) Syntactic categories are capitalized while all literals are in lowercase.

(4) { } means optional.

(5) [] means one of the enclosed choices must appear; choices are separated by a ';' (when used with '{}' one of the choices may optionally appear).

(6) <> means one or more of the enclosed can appear, separated by spaces with optional comma and an optional 'and' at the end.

(7) <<>> means one or more of the enclosed can appear, vertically appended.

(8) * * encloses a 'meta' description of a syntactic category (to informally explain it).

```
SCHEMA <==
  << CLASS >>

CLASS <==
  < CLASS_NAME >
    { description: CLASS_DESCRIPTION }
    {[ BASE_CLASS_FEATURES; INTERCLASS_CONNECTION ]}
    { MEMBER_ATTRIBUTES }
    { CLASS_ATTRIBUTES }

CLASS_NAME <==
  * string of capitals possibly including special characters *

CLASS_DESCRIPTION <==
  * string *

BASE_CLASS_FEATURES <==
  {[ duplicates allowed; duplicates not allowed ]}
  {<< IDENTIFIERS >>}

IDENTIFIERS <==
  [ ATTRIBUTE_NAME; ATTRIBUTE_NAME + IDENTIFIERS ]
```

```
MEMBER_ATTRIBUTES <==
  member attributes:
    << MEMBER_ATTRIBUTE >>

CLASS_ATTRIBUTES <==
  class attributes:
    << CLASS_ATTRIBUTE >>

INTERCLASS CONNECTION <==
    [ SUBCLASS; GROUPING_CLASS ]

SUBCLASS <==
  subclass of CLASS_NAME where SUBCLASS_PREDICATE

GROUPING <==
  [ grouping of CLASS_NAME on common value of < ATTRIBUTE_NAME >
    { groups defined as classes are < CLASS_NAME >};
    grouping of CLASS_NAME consisting of classes < CLASS_NAME >;
    grouping of CLASS_NAME as specified ]

SUBCLASS_PREDICATE <==
  [ ATTRIBUTE_PREDICATE;
    specified;
    is in CLASS_NAME and is in CLASS_NAME;
    is not in CLASS_NAME;
    is in CLASS_NAME or is in CLASS_NAME;
    is a value of ATTRIBUTE_NAME of CLASS_NAME;
    format is FORMAT ]

ATTRIBUTE_PREDICATE <==
  [ SIMPLE_PREDICATE; ( ATTRIBUTE_PREDICATE );
    not ATTRIBUTE_PREDICATE;
    ATTRIBUTE_PREDICATE and ATTRIBUTE_PREDICATE;
    ATTRIBUTE_PREDICATE or  ATTRIBUTE_PREDICATE ]

SIMPLE_PREDICATE <==
  [ MAPPING SCALER_COMPARATOR [ CONSTANT; MAPPING ];
    MAPPING SET_COMPARATOR [ CONSTANT; CLASS_NAME; MAPPING ]]

MAPPING <==
  [ ATTRIBUTE_NAME; MAPPING.ATTRIBUTE_NAME ]

SCALER_COMPARATOR <==
  [ EQUAL_COMPARATOR; >; >=; <; =<; ]

EQUAL_COMPARATOR <==
  [ =; >< ]

SET_COMPARATOR <==
  [ is ( properly ) contained in; ( properly ) contains ]
```

```
CONSTANT <==
  * a string or number constant *

FORMAT <==
  * a name class definition pattern *
  (see [16])

MEMBER_ATTRIBUTE <==
  < ATTRIBUTE_NAME >
    { ATTRIBUTE_DESCRIPTION }
    value class: CLASS_NAME
    { inverse: ATTRIBUTE_NAME }
    {[ match: ATTRIBUTE_NAME of CLASS_NAME on ATTRIBUTE_NAME;
       derivation: MEMBER_ATTRIBUTE_DERIVATION ]}
    { single valued; multivalued { with size between CONSTANT and
      CONSTANT }}
    { may not be null }
    { not changeable }
    { exhausts value class }
    { no overlap in values }

CLASS_ATTRIBUTE <==
  < ATTRIBUTE_NAME >
    { ATTRIBUTE_DESCRIPTION }
    value class: CLASS_NAME
    { derivation: CLASS_ATTRIBUTE_DERIVATION }
    { single valued; multivalued { with size between CONSTANT and
      CONSTANT }}
    { may not be null }
    { not changeable }

ATTRIBUTE_NAME <==
  * string of lowercase letters beginning with a capital and possibly
    including special characters *

ATTRIBUTE_DESCRIPTION <==
  * string *

MEMBER_ATTRIBUTE_DERIVATION <==
  [ INTER_ATTRIBUTE_DERIVATION; MEMBER_SPECIFIC_DERIVATION ]

CLASS_ATTRIBUTE_DERIVATION <==
  [ INTER_ATTRIBUTE_DERIVATION; CLASS_SPECIFIC_DERIVATION ]
```

```
INTER_ATTRIBUTE_DERIVATION <==
  [ same as MAPPING;
    subvalue of MAPPING where [ is in CLASS_NAME;
      ATTRIBUTE_PREDICATE ];
    where [ is in MAPPING and is in MAPPING; is in MAPPING or is in
      MAPPING; is in MAPPING and is not in MAPPING ];
    = MAPPING_EXPRESSION;
    [ maximum; minimum; average; sum ] of MAPPING;
      number of { unique } members in MAPPING ]

MEMBER_SPECIFIC_DERIVATION <==
  [ ordering by [ increasing; decreasing ] < MAPPING >
    { within < MAPPING >};
    if in CLASS_NAME;
    [ up to CONSTANT; all ] levels of values of ATTRIBUTE_NAME;
        contents ]

CLASS_SPECIFIC_DERIVATION <==
  [ number of { unique } members in this class;
    [ maximum; minimum; average; sum ] of ATTRIBUTE_NAME over members
        of this class ]

MAPPING_EXPRESSION <==
  [ MAPPING; < MAPPING >; MAPPING NUMBER_OPERATOR MAPPING ]

NUMBER_OPERATOR <==
  [ +; -; *; /; ! ]
```

APPENDIX B SDM Tanker Monitoring Application

(Source: Hammer and McLeod, [12, pp. 376-381])

```
SHIPS
  description:  all ships with potentially hazardous cargoes that may
                enter U.S. coastal waters

  member attributes:
    Name
      value class: SHIP_NAMES
    Hull_number
      value class: HULL_NUMBERS
      may not be null
      not changeable
    Type
      description: the kind of ship, for example, merchant or fishing
      value class: SHIP_TYPE_NAMES
    Country_of_registry
      value class: COUNTRIES
      inverse    : Ships_registered_here
    Name_of_home_port
      value class: PORT_NAMES
    Cargo_types
      description: the type(s) of cargo the ship can carry
      value class: CARGO_TYPE_NAMES
      multivalued
    Captain
      description: the current captain of the ship
      value class: OFFICERS
      match      : Officer of ASSIGNMENTS on Ship
    Engines
      value class: ENGINES
      multivalued with size between 0 and 10
      exhausts value class
      no overlap in values
    Incidents_involved_in
      value class: INCIDENTS
      inverse    : Involved_ship
      multivalued

  identifiers:
    Name
    Hull_number
```

```
INSPECTIONS
  description:  inspections of oil tankers

  member attributes:
    Tanker
      description: the tanker inspected
      value class: OIL_TANKERS
      inverse    : Inspections
    Date
      value class: DATES
    Order_for_tanker
      description: the ordering of the inspections for a tanker with
                   the most recent inspection having value 1
      value class: INTEGERS
      derivation : order by decreasing Date within Tanker

  class attributes:
    Number
      description: the number of inspections in the database
      value class: INTEGERS
      derivation : number of members in this class

  identifiers:
    Tanker + Date


COUNTRIES
  description:  countries of registry for ships

  member attributes:
    Name
      value class: COUNTRY_NAMES
    Ships_registered_here
      value class: SHIPS
      inverse    : Country_of_registry
      multivalued

  identifiers
    Name
```

```
OFFICERS
  description:  all certified officers of ships

  member attributes:
    Name
      value class: PERSON_NAMES
    Country_of_license
      value class: COUNTRIES
    Date_commissioned
      value class: DATES
    Seniority
      value class: INTEGERS
      derivation : order by Date_commissioned
    Commander
      description: the officer in direct command of this officer
      value class: OFFICERS
    Superiors
      value class: OFFICERS
      derivation : all level of values of Commander
      inverse    : Subordinates
      multivalued
    Subordinates
      value class: OFFICERS
      inverse    : Superiors
      multivalued
    Contacts
      value class: OFFICERS
      derivation : where is in Superiors or is in Subordinates

  identifiers:
    Name


ENGINES
  description:  ship engines

  member attributes:
    Serial_number
      value class: ENGINE_SERIAL_NUMBERS
    Kind_of_engine
      value class: ENGINE_TYPE_NAMES

identifiers:
  Serial_number
```

```
INCIDENTS
  description:  accidents involving ships

  member attributes:
    Involved_ship
      value class: SHIPS
      inverse    : Incidents_involved_in
    Date
      value class: DATES
    Description
      description: textual explanation of the accident
      value class: INCIDENT_DESCRIPTIONS
    Involved_captain
      value class: OFFICERS

  identifiers:
    Involved_ship + Date + Description
```

```
ASSIGNMENTS
  description:  assignments of captains to ships

  member attributes:
    Officer
      value class: OFFICERS
    Ship
      value class: SHIPS

  identifiers:
    Officer + Ship
```

```
OIL_TANKERS
  description:  oil-carrying ships
  interclass connection:  subclass of SHIPS where Cargo_types
    contains 'oil'

  member attributes:
    Hull_type
      description: specification of single or double hull
      value class: HULL_TYPE_NAMES
    Is_tanker_banned
      value class: YES/NO
      derivation : if in BANNED_SHIPS
    Inspections
      value class: INSPECTIONS
      inverse    : Tanker
      multivalued
    Number_of_times_inspected
      value class: INTEGERS
      derivation : number of unique members in Inspections
    Last_inspection
      value class: MOST_RECENT_INSPECTIONS
      inverse    : Tanker
    Last_two_inspections
      value class: INSPECTIONS
      derivation : subvalue of Inspections where Order_for_tanker =< 2
      multivalued
    Date_last_examined
      value class: DATES
      derivation : same as Last_inspection.Date
    Oil_spills_involved_in
      value class: INCIDENTS
      derivation : subvalue of Incidents_involved_in where is in
                   OIL_SPILLS
      multivalued

class attributes:
  Absolute_top_legal_speed
    value class: KNOTS
  Top_legal_speed_in_miles_per_hour
    value class: MILES_PER_HOUR
    derivation : = Absolute_top_legal_speed/1.1


RURITANIAN_SHIPS
  interclass connection:  subclass of SHIPS where
    Country.Name = 'Ruritania'


RURITANIAN_OIL_TANKERS
  interclass connection:  subclass of OIL_TANKERS where
    Country.Name = 'Ruritania'
```

```
MERCHANT_SHIPS
  interclass connection:  subclass of SHIPS where Type = 'merchant'

  member attributes:
    Cargo_types
      value class: MERCHANT_CARGO_TYPE_NAMES


OIL_SPILLS
  interclass connection:  subclass of INCIDENTS where
    Description = 'oil spill'

  member attributes:
    Amount_spilled
      value class: GALLONS
    Severity
      derivation : = Amount_spilled/100,000

  class attributes:
    Total_spilled
      value class: GALLONS
      derivation : sum of Amount_spilled over members of this class


MOST_RECENT_INSPECTIONS
  interclass connection:  subclass of INSPECTIONS where
    Order_for_tanker = 1


DANGEROUS_CAPTAINS
  description:  captains who have been involved in an accident
  interclass connection:  subclass of OFFICERS where IS A VALUE OF
    Involved_captain of INCIDENTS


BANNED_SHIPS
  description:  ships banned from U.S. coastal waters
  interclass connection:  subclass of SHIPS where specified

  member attributes:
    Date_banned
      value class: DATES


OIL_TANKERS_REQUIRING_INSPECTION
  interclass connection:  subclass of OIL_TANKERS where specified


BANNED_OIL_TANKERS
  interclass connection:  subclass of SHIPS where is in BANNED_SHIPS
    and is in OIL_TANKERS
```

SAFE_SHIPS
  description: ships that are considered good risks
  interclass connection: subclass of SHIPS where is not in
    BANNED_SHIPS

SHIPS_TO_BE_MONITORED
  description: ships that are considered bad risks
  interclass connection: subclass of SHIPS where is in BANNED_SHIPS or
    is in OIL_TANKERS_REQUIRING_INSPECTION

SHIP_TYPES
  description: types of ships
  interclass connection: grouping of SHIPS on common value of Type
    groups defined as classes are MERCHANT_SHIPS

  member attributes:
    Instances
      description: the instances of the type of ship
      value class: SHIPS
      derivation : same as Contents
      multivalued
    Number_of_ships_of_this_type
      value class: INTEGERS
      derivation : number of members in Contents

CARGO_TYPE_GROUPS
  interclass connection: grouping of SHIPS on common value of
    Cargo_types

TYPES_OF_HAZARDOUS_SHIPS
  interclass connection: grouping of SHIPS consisting of classes
    BANNED_SHIPS, BANNED_OIL_TANKERS, SHIPS_TO_BE_MONITORED

CONVOYS
  interclass connection: grouping of SHIPS as specified

  member attributes:
    Oil_tanker_constituents
      description: the oil tankers that are in the convoy (if any)
      value class: SHIPS
      derivation : subvalue of Contents where is in OIL_TANKERS
      multivalued

CARGO_TYPE_NAMES
  description: the types of cargo
  interclass connection: subclass of STRINGS

MERCHANT_CARGO_TYPE_NAMES
  interclass connection: subclass of CARGO_TYPE_NAMES where specified

COUNTRY_NAMES
  interclass connection: subclass of STRINGS where specified

ENGINE_SERIAL_NUMBERS
  interclass connection: subclass of STRINGS where format is
                          'H'
                          number where integer and >= 1 and =< 999
                          '-'
                          number where integer and >= 0 and =< 999999

DATES
  description: calendar dates in the range '1/1/85' to '12/31/89'
  interclass connection: subclass of STRINGS where format is
                          month:number where >= 1 and =< 12
                          '/'
                          day :number where integer and >= 1   and =< 31
                          year :number where integer and >= 1970 and =< 2000
                          where if (month = 4 or = 5 or = 9 or = 11) then day
                          >= 30) and (if month = 2 then day =< 29)
                          ordering by year, month, day

ENGINE_TYPE_NAMES
  interclass connection: subclass of STRINGS where specified

GALLONS
  interclass connection: subclass of STRINGS where format is number
    where integer

HULL_NUMBERS
  interclass connection: subclass of STRINGS where format is number
    where integer

HULL_TYPE_NAMES
  description: single or double
  interclass connection: subclass of STRINGS where specified

INCIDENT_DESCRIPTIONS
  description: textual description of an accident
  interclass connection: subclass of STRINGS

KNOTS
  interclass connection: subclass of STRINGS where format is number
    where integer

MILES_PER_HOUR
  interclass connection: subclass of STRINGS where format is number
    where integer

PORT_NAMES
  interclass connection: subclass of STRINGS

PERSON_NAMES
  interclass connection: subclass of STRINGS

SHIP_NAMES
  interclass connection: subclass of STRINGS

SHIP_TYPE_NAMES
  description: the names of the ship types, for example, merchant
  interclass connection: subclass of STRINGS where specified

APPENDIX C English Rule Interpretation

Class Generic Rules

Rule_1 — **The class name must be unique and composed of uppercase letters and underscores. The class name will be stored as lower case and converted to uppercase for display.**

Rule_2 — A class name can have synonyms (optional). Each synonym must be unique, composed of uppercase letters and underscores. The synonym will be stored as lower case and converted to uppercase for display.

Rule_3 — A class name can have descriptive text (optional) describing the meaning and contents of the class. The text is a string of characters.

Base Class Rules

Rule_4 — A class name must be a base class or nonbase class.

Rule_4A — A base class must have at least one (1) logical key. A logical key is composed of member attributes, and each member attribute must exist within the base class. A logical key can have duplicate or unique values. Its default is duplicate values. A member attribute cannot be repeated within a logical key.

Nonbase Class Rules

Rule_6 — A nonbase class has one interclass connection, either a sub-class or grouping class connection. A subclass name must exist and cannot be equal to the nonbase class name.

Rule_6A	**A subclass can have one of one of the following predicates (optional):** **attribute-defined** **difference** **existence** **format** **intersection** **none** **union** **where-specified**
Rule_6A1	An attribute-defined predicate defines a subclass S, composed of members of C, based on a member attribute equation: equality contains
Rule_6A2	A difference predicate defines a subclass S composed of members from class C that do not belong to class D. Class D must exist and must be a subclass of C. Class name S cannot be equal to class name C, D, or E.
Rule_6A3	An existence predicate defines a subclass S composed of members from class C, based on a member attribute of C. The member attribute must exist in class C. Class name S cannot be equal to class name C.
Rule_6A4	A format predicate is free form textual description.
Rule_6A5	An intersection predicate defines a subclass S composed of members from class C that belong to class D and E. Class D and E must exist and both must be a subclass of C. Class name S cannot be equal to class name C, D, or E.
Rule_6A6	No predicate will be defined.
Rule_6A7	A union predicate defines a subclass S composed of members from class C that belong to class D or E. Class D and E must exist and both must be a subclass of C. Class name S cannot be equal to class name C, D, or E.

Rule_6A8	A whose-specified predicate defines a subclass S, composed of members from class C, manually controlled.
Rule_6A1A	Equality defines an equation based on equal (=), not equal (><), greater than or equal (>=), greater than (>), less than (<), less than or equal (=<). The member attribute must exist, the comparator must be one of the above, and either a constant or another member attribute. The member attributes cannot be equal, and the second member attribute must exist. For example cargo_type = 'oil'. (Member attribute was not implemented).
Rule_6A1B	Contains defines an equation based on properly contained, contained, properly contains, contains. The member attribute must exist, the set comparator must be one of the above, and either a constant, member attribute, or class name. The member attributes cannot be equal, and the second member attribute must exist. For example cargo_type = 'oil'. The class name must exist, and cannot be equal to the class being created. (Member attribute and class name were not implemented).
Rule_7	A member attribute name must be unique within the class and all subclasses, composed of letters and underscores. The member attribute name will be stored as lower case and the first letter of the name will be converted to uppercase for display.
Rule_7A	A member attribute can exhaust its value class and valid values are 'Yes' or 'No'. The default is 'No'.
Rule_7B	A member attribute cannot be changed once a value is assigned. The value can be changed only to correct an error. Valid values are 'Yes' or 'No'. The default is 'No'.
Rule_7C	A member attribute can be mandatory. Valid values are 'Yes' or 'No'. The default is 'No'. If a member attribute has no value class (Rule_8), the member attribute value cannot be mandatory.

Rule_7D	**A member attribute can be single valued or multivalued. Valid values are 'S', single valued; or 'M', multivalued. The default is single valued.**
Rule_7D1	A multivalued attribute can have one of the following (optional): nonoverlapping value restriction
Rule_7D1A	Nonoverlapping means each member of the value class of the attribute is used only once. Valid values are 'Yes' or 'No'. Default is 'No'.
Rule_7D1B	Value restriction is a range of values a member attribute can contain. The lower range is an integer. The upper range is an integer. The upper range integer must be greater than the lower range integer.
Rule_7F	A member attribute can have descriptive text (optional) describing the meaning and contents of the member attribute. The text is a string of characters.
Rule_8	A member attribute can have a value class (optional). The value class name must exist. If there is no value class, the member attribute cannot be mandatory (Rule_7C). (The no value class option was not implemented).

APPENDIX D Prolog Implementation of SDM

```
code=4000

include "tdoms.pro"

database - sdmdb
  base_class          (string,stringlist)
  class_def           (string,string)
  class_list          (stringlist)
  class_members       (string,stringlist)
  class_syn           (string,stringlist)
  member_attr         (string,string,string,string,stringlist)
  non_base            (string,string,string,string,string)
  stnd_class          (stringlist)

include "tpreds.pro"
include "lineinp.pro"
include "menu2.pro"
include "status.pro"

predicates
  asc_order           (string,string)
  append              (stringlist,stringlist,stringlist)
  base_list           (string,string)
  create_class        (string,string)
  create_def          (string,string)
  delete              (string,stringlist,stringlist)
  get_class           (symbol,string)
  get_class_members   (string,stringlist)
  go
  init
  init_stnd_class     (stringlist)
  insert              (string,stringlist,stringlist)
  insert_sort         (stringlist,stringlist)
  makenblist          (string,stringlist)
  member              (string,stringlist)
  menuchoice          (integer)
  pause
  print_class_members (stringlist)
  print_list          (stringlist)
  print_member        (string)
  print_member_attr_list (stringlist)
  print_member_attr   (string)
  print_member_def    (string)
  print_non_base      (string,string,string)
  rule1               (string)
  rule2               (string,string)
  rule3               (string)
  rule6               (string,string,string,string,string)
  rule6a              (integer,string,string,string,string,string)
  rule6a1             (string,string)
  rule7               (string)
  rule7a              (string,string)
```

```
  rule7b              (stringlist)
  rule7b1             (integer,stringlist)
  rule7b2             (integer,stringlist)
  selection           (integer,stringlist,string)
  update_class_members (string,string)
  valid_name          (string)

goal
  retractall(_,sdmdb),
  makewindow(1,23,14," Semantic Data Model ",0,0,24,80),
  go.


clauses

go:-
  init,
  repeat,
  menu (8,27,31,11,
       [" Class Add"," Synonym Add"," Class Definition",
        " Member Attributes"," Non-base Add"," List Classes",
        " List Synonyms"," List Class Definition",
        " List Member Attributes "," List Nonbase Classes",
        " Exit"],
        " Main Menu ",1,Choice),
  menuchoice(Choice),
  clearwindow,
  Choice=11,
  exit.

init:-
  assertz(class_list([])),
  assertz(stnd_class(["integers",
    "numbers","reals","strings","yes/no"])),
  stnd_class(SLIST),
  init_stnd_class(SLIST),
  makestatus(112,"").

init_stnd_class([]).
init_stnd_class([X|TAIL]):-
  init_stnd_class(TAIL),
  assertz(base_class(X,[])).
```

```
menuchoice(1):- /* create class*/
  changestatus(""),
  lineinput(10,10,45,7,7,"Class name: ","",INPUT),
  upper_lower(INPUT,CLASS),
  valid_name(CLASS),
  rule1(CLASS),
  menu(15,10,31,11,["No","Yes"]," Create definition? ",1,S),
  selection(S,["n","y"],DEFOPT),
  create_def(DEFOPT,DEF),
  menu(15,30,31,11,["Base Class","Non_Base Class"],
               " Select Class Type ",1,S2),
  selection(S2,["b","n"],CLASSOPT),
  create_class(CLASSOPT,CLASS),
  class_list(LIST),
  retract(class_list(LIST)),
  insert_sort([CLASS|LIST],SORTED_LIST),
  assertz(class_list(SORTED_LIST)),
  assertz(class_def(CLASS,DEF)),
  changestatus("Class added").

menuchoice(2):- /* create synonyms*/
  get_class(all,CLASS),
  upper_lower(UPPER,CLASS),
  concat(UPPER," synonym: ",MENUTEXT),
  lineinput(10,10,45,7,7,MENUTEXT,"",INPUT),
  upper_lower(INPUT,SYN),
  valid_name(SYN),
  rule2(CLASS,SYN),
  class_syn(CLASS,SYNLIST),
  retract(class_syn(CLASS,SYNLIST)),
  insert_sort([SYN|SYNLIST],SORTLIST),
  assertz(class_syn(CLASS,SORTLIST)),
  changestatus("Synonym added"),
  !.

menuchoice(3):- /* class definition*/
  get_class(all,CLASS),
  rule3(CLASS).

menuchoice(4):- /* member attributes*/
  get_class(all,CLASS),
  rule7(CLASS).
/*
menuchoice(5):- /* non-base classes*/
  get_class(all,NBCLASS),
  rule6.
*/
menuchoice(6):- /* no classes exist*/
  class_list([]),
  beep,
  changestatus("No Classes have been defined"),!.
```

```
menuchoice(6):- /* list all classes*/
  class_list(LIST),
  print_list(LIST),
  pause,
  changestatus(""),!.

menuchoice(7):- /* list synonyms*/
  class_syn(CLASS,LIST),
  upper_lower(UPPER,CLASS),
  write("Class ",UPPER," synonym(s): "),
  print_list(LIST),nl,
  pause,
  fail.

menuchoice(7):- /* no synonyms exist*/
  not(class_syn(_,_)),
  beep,
  changestatus("No Synonyms have been defined").

menuchoice(8):- /* list class definition */
  class_def(Class,Text),
  Text >< "",
  upper_lower(Upper,Class),
  write("Class ",Upper," definition: "),nl,
  write(Text),nl,
  pause,
  fail.

menuchoice(8):- /* no definitions exist*/
  not(class_def(_,_)),
  beep,
  changestatus("No Class definitions have been defined").

menuchoice(9):- /* list member attributes*/
  class_list(LIST),
  print_class_members(LIST).

menuchoice(10):-/* list non-base class          */
  non_base(CLASS,SUBCLASS,TYPE,FORM,FORM2),
  upper_lower(UPPER,CLASS),
  write(UPPER),nl,
  upper_lower(UPPER2,SUBCLASS),
  write("  interclass connection: subclass of ",UPPER2),
  print_non_base(TYPE,FORM,FORM2),nl,
  pause,
  fail.

menuchoice(_).  /* menuchoice will always succeed */
```

```
valid_name(X):- /* valid name*/
  not(isname(X)),
  changestatus("Invalid name"),
  !,fail.

valid_name(_).

rule1(CLASS):-  /* class cannot exist*/
  class_list(CLIST),
  stnd_class(STCLASS),
  append(CLIST,STCLASS,LIST),
  member(CLASS,LIST),
  changestatus("Class name already exists"),
  !,fail.

rule1(_).

rule2(_,SYN):-  /* syn cannot exist */
  class_syn(CLASS,CLIST),
  stnd_class(STCLASS),
  append(CLIST,STCLASS,LIST),
  member(SYN,LIST),
  upper_lower(UPPER,CLASS),
  concat("Synonym exists for class ",UPPER,TEXT2),
  changestatus(TEXT2),
  !,fail.

rule2(_,SYN):-  /* syn cannot exist as a class*/
  class_list(LIST),
  member(SYN,LIST),
  changestatus("Synonym exists as a class"),
  !,fail.

rule2(CLASS,_):-/* create dummy synonym*/
  not(class_syn(CLASS,_)),
  assertz(class_syn(CLASS,[])),
  !.

rule2(_,_).


rule3(Class):-  /* create class definition*/
  class_def(Class,Textin),
  edit(Textin,Textout),
  Textin >< Textout,
  retract(class_def(Class,_)),
  !,
  Textout >< "",
  assertz(class_def(Class,Textout)).
```

```
rule3(Class):-
  not(class_def(Class,_)),
  edit("",Textout),
  Textout >< "",
  assertz(class_def(Class,Textout)).


rule6(NBCLASS,SUBCLASS,TYPE,FORM,FORM2):-/* non-base class*/
  get_class(sub,SUBCLASS),
/*  delete(NBCLASS,SUBCLASSL,SUBCLASSL2),  */
  menu(13,10,31,11,["Attribute defined ","Difference",
                    "Existence","Format","Intersection",
                    "None","Union","Where Specified"],
                    " Select Subclass Type ",1,S),
  rule6a(S,NBCLASS,SUBCLASS,TYPE,FORM,FORM2).

rule6a(1,_,SUBCL,NBTYPE,FORM,FORM2):-  /* non-base type = attribute */
  get_class_members(SUBCL,MLIST),
  menu(13,10,31,11,MLIST," Select Attribute ",1,S),
  selection(S,MLIST,FORM),
  menu(13,10,31,11,["equal","not equal","less than",
                    "less than or equal","greater than",
                    "greate than or equal","contains",
                    "properly contains","contained in",
                    "is properly contained in "],
                    " Select Comparator ",1,S2),
   selection(S2,["aeq","ane","alt","ale","agt","age","aco","apc",
          "acd","apd"],NBTYPE),
  menu(13,10,31,11,["Constant","Attribute"]," Select 2nd type ",1,
       S3),
  selection(S3,["C","A"],MAPTYPE2),
  rule6a1(MAPTYPE2,FORM2),
  !.

rule6a(2,_,SUBCL,NBTYPE,FORM,FORM2):-/* non-base type = difference*/
  NBTYPE = "d",
  FORM2  = "",
  makenblist(SUBCL,NBLIST),
/*  delete(NBCL,NBLIST,NBLIST2),/* change nblist = nblist3*/
  delete(SUBCL,NBLIST2,NBLIST3),
*/  menu(15,20,31,11,NBLIST," Select Difference Sub_class ",1,S),
  selection(S,NBLIST,FORM),
  !.

rule6a(3,_,_,NBTYPE,FORM,FORM2):-/* non-base type = existence*/
  NBTYPE = "e",
  FORM   = "existence",
  FORM2  = "",
  !.
```

```
rule6a(4,_,_,NBTYPE,FORM,FORM2):-/* non-base type = format*/
  NBTYPE = "f",
  FORM2  = "",
  edit("",FORM),
  !.

rule6a(5,_,SUBCL,NBTYPE,FORM,FORM2):-/* non-base type = intersection */
  NBTYPE = "i",
  makenblist(SUBCL,NBLIST),
/*  delete(NBCL,NBLIST,NBLIST2),*/
  menu(15,20,31,11,NBLIST,
    " Intersection: select 1st Sub_class ",1,S1),
  selection(S1,NBLIST,FORM),
  delete(FORM,NBLIST,NBLIST2),
  menu(15,20,31,11,NBLIST2," Select 2nd Sub_class ",1,S2),
  selection(S2,NBLIST2,FORM2),
  !.

rule6a(6,_,_,NBTYPE,FORM,FORM2):-/* non-base type = none*/
  NBTYPE = "n",
  FORM   = "",
  FORM2  = "",
  !.

rule6a(7,_,SUBCL,NBTYPE,FORM,FORM2):-/* non-base type = union */
  NBTYPE = "u",
  makenblist(SUBCL,NBLIST),
  menu(15,20,31,11,NBLIST," Union: select 1st Sub_class ",1,S1),
  selection(S1,NBLIST,FORM),
  delete(FORM,NBLIST,NBLIST2),
  menu(15,20,31,11,NBLIST2," Select 2nd Sub_class ",1,S2),
  selection(S2,NBLIST2,FORM2),
  !.

rule6a(8,_,_,NBTYPE,FORM,FORM2):-/* non-base type = specified*/
  NBTYPE = "s",
  FORM   = "",
  FORM2  = "",
  !.
```

```
rule7(CLASS):-  /* create member attributes*/
  upper_lower(UPPER,CLASS),
  concat(UPPER," member name: ",MTEXT),
  lineinput(10,10,45,7,7,MTEXT,"",Input),
  upper_lower(INPUT,MEMBER),
  valid_name(MEMBER),
  rule7a(CLASS,MEMBER),
  get_class(value,VCLASS),
  rule7b(ATTRLIST),
  menu(8,27,31,11,["No","Yes"]," Create definition? ",1,S),
  selection(S,["n","y"],DEFOPT),
  create_def(DEFOPT,DEF),
  asserta(member_attr(CLASS,MEMBER,VCLASS,DEF,ATTRLIST)),
  update_class_members(CLASS,MEMBER),
  changestatus("Member attribute added"),
  !.


rule6a1("C",CONSTANT):-
  lineinput(10,10,45,7,7,"Contant is: ","",CONSTANT), !.

rule6a1("A",_):-
  changestatus("Function not available"),
  fail.

rule7a(C,M):-   /* member cannot exist*/
  member_attr(C,M,_,_,_),
  changestatus("Member attribute already exists"),
  !,fail.

rule7a(_,_).

rule7b(ATTR_CHARACTERISTICS):-
  menu_repeat(10,10,31,11,["Mandatory","Optional"],
                  " Attribute is ",2,S1),
  selection(S1,["M","O"],MANDOPT),
  append([],[MANDOPT],OPTLIST1),
  menu_repeat(10,40,31,11,["Changeable","Not Changeable"],
                  " Attribute is ",1,S2),
  selection(S2,["C","NC"],CHGOPT),
  append(OPTLIST1,[CHGOPT],OPTLIST2),
  menu_repeat(15,10,31,11,["No","Yes"],
                  " Exhausts Value Class ",1,S3),
selection(S3,["N","E"],EXHOPT),
append(OPTLIST2,[EXHOPT],OPTLIST3),
menu_repeat(15,40,31,11,["Single Value","Mult-Value"],
                  " Attribute is ",1,S4),
rule7b1(S4,SMLIST),
append(OPTLIST3,SMLIST,ATTR_CHARACTERISTICS),
removewindow,removewindow,removewindow,removewindow,
!.
```

```
rule7b1(1,["S"]).
rule7b1(2,MLIST):-
  menu_repeat(12,10,31,11,["No","Yes"]," Overlap in Values ",1,S1),
  selection(S1,["N","O"],OVEROPT),
  append([],[OVEROPT],MLIST1),
  menu_repeat(12,40,31,11,["No","Yes"]," Range Limitation ",1,S2),
  rule7b2(S2,RANGE),
  append(MLIST1,RANGE,MLIST),
  removewindow,
  removewindow.

rule7b2(1,[]).
rule7b2(2,RANGE):-
  lineinput(16,40,15,7,7,"Low Range: ","",NUMSTART),
  lineinput(16,60,15,7,7,"High Range: ","",NUMEND),
  append([NUMSTART,NUMEND],[],RANGE).

create_def("y",Textout):-/* invoke Prolog editor */
  edit("",Textout),
  clearwindow,
  !.

create_def(_,Text):-/* if editor aborted set null */
  Text="".

create_class("b",BCLASS):-/* create base class*/
  assertz(base_class(BCLASS,[])), !.

create_class("n",NBCLASS):-/* create non_base class*/
  rule6(NBCLASS,SUBCLASS,TYPE,FORM,FORM2),
  assertz(non_base(NBCLASS,SUBCLASS,TYPE,FORM,FORM2)),
  base_list(NBCLASS,SUBCLASS).
```

```
print_non_base("d",TEXT,_):-/* print difference sub_class */
  upper_lower(UPPER,TEXT),
  write(" where is not in ",UPPER),!.

print_non_base("f",TEXT,_):-/* print format sub_class*/
  write(" where format is ",TEXT),!.


print_non_base("n",_,_):- !./* print none sub_class*/

print_non_base("s",_,_):-/* print user controlled sub*/
  write(" where specfied"),!.

print_non_base("u",TEXT,TEXT2):-/* print intersection sub_class */
  upper_lower(UPPER,TEXT),
  write(" where is in ",UPPER),nl,
  upper_lower(UPPER2,TEXT2),
  write(" and is in ",UPPER2),!.

print_non_base("aeq",TEXT,TEXT2):-
   frontstr(1,TEXT,FIRST,REST),
   upper_lower(FUPPER,FIRST),
   concat(FUPPER,REST,MATTR),
   concat("'",TEXT2,CONSTANT),
   concat(CONSTANT,"'",CONSTANT2),
   write(" where ",MATTR," = ",CONSTANT2),!.

print_class_members([]).* print member attributes*/
print_class_members([Class|Tail]):-
   print_class_members(Tail),
   upper_lower(Upper,Class),
   write(Upper),
   write("\n member attributes: "),
   print_member(Class),
   changestatus("Press ENTER to return"),
   pause.

print_member(C):-/* print member details   */
   member_attr(C,M,V,D,A),
   frontstr(1,M,FIRST,REST),
   upper_lower(FUPPER,FIRST),
   concat(FUPPER,REST,MATTR),
   upper_lower(VCLASS,V),
   write("\n   ",MATTR),
   print_member_def(D),
   write("\n     value class: ",VCLASS),
   print_member_attr_list(A),
   pause,
   fail.

/*print_member(_).*/
```

```
print_member_attr_list([]).
print_member_attr_list([H|TAIL]):-
  print_member_attr(H),!,
  print_member_attr_list(TAIL).


print_member_attr("M"):-
  write("\n     may not be null").

print_member_attr("NC"):-
  write("\n     not changeable").

print_member_attr("E"):-
  write("\n     exhausts value class").

print_member_attr(_):- !.


print_member_def(D):-
  D >< "",
  write("\n     description: ",D),!.

print_member_def(_).
```

```
get_class(all,C):-/* display menu of classes*/
  changestatus(""),
  class_list(CHOICELIST),
  menu(8,27,31,11,CHOICELIST," Select Class from menu ",1,S),
  selection(S,CHOICELIST,C),!.

get_class(all,_):-
  class_list([]),
  beep,
  changestatus("No Classes have been defined"),
  !,fail.


get_class(sub,C):-/* display menu of sub-classes*/
  changestatus(""),
  stnd_class(STCLASS),
  class_list(CLIST),
  append(CLIST,STCLASS,CLASSLIST),
  insert_sort(CLASSLIST,SORTLIST),
  menu(8,27,31,11,SORTLIST," Select Subclass ",1,S),
  selection(S,SORTLIST,C),
  !.

get_class(value,C):-/* display menu of value classes*/
  changestatus(""),
  class_list(CLIST),
  stnd_class(SLIST),
  append(CLIST,SLIST,CHOICELIST),
  insert_sort(CHOICELIST,SORTLIST),
  menu(8,27,31,11,SORTLIST," Select Value Class ",1,S),
  selection(S,SORTLIST,C),
  !.

get_class_members(C,ML):-
  class_members(C,ML), !.

get_class_members(_,_):-
  changestatus("No members exist for non_base class "),
  pause,
  fail.
```

```
append([],LIST,LIST)./* append two lists into one*/
append([X|L1],LIST2,[X|L3]):-
  append(L1,LIST2,L3).


base_list(NB,SC):-/* add subclass to base_class*/
  base_class(SC,LIST),
  not(member(NB,LIST)),
  retract(base_class(C,LIST)),
  insert_sort([NB|LIST],SORTLIST),
  assertz(base_class(C,SORTLIST)), !.

base_list(NB,SC):-/* locate base_class of sub_class */
  base_class(C,LIST),
  member(SC,LIST),
  retract(base_class(C,LIST)),
  insert_sort([NB|LIST],SORTLIST),
  assertz(base_class(C,SORTLIST)), !.

delete(_,[],[])./* delete X from list*/
delete(X,[X|L1],L1):- !.
delete(X,[Y|L1],[Y|L2]):- !,
  delete(X,L1,L2).

insert_sort([],[])./* sort routine*/
insert_sort([X|TAIL],SORTED_LIST):-
  insert_sort(TAIL,SORTED_TAIL),
  insert(X,SORTED_TAIL,SORTED_LIST).

insert(X,[Y|SORTED_LIST],[Y|SORTED_LIST1]):-
  asc_order(X,Y), !,
  insert(X,SORTED_LIST,SORTED_LIST1).

insert(X,SORTED_LIST,[X|SORTED_LIST]).

asc_order(X,Y):- X>Y.


makenblist(SC,NBLIST):-/* make non-base selection list */
  base_class(SC,NBLIST), !.

makenblist(SC,NBLIST):-
  base_class(_,LIST),
  member(SC,LIST),
  delete(SC,LIST,NBLIST), !.


member(X,[X|_])./* is X a found in LIST*/
member(X,[_|Y]):- member(X,Y).
```

```
pause:-          /* pause*/
  changestatus("Press ENTER to continue"),
  readln(_).

print_list([]):- !./* print a member list*/
print_list([X|Y]):-
  upper_lower(UPPER,X),
  write("\n     ",UPPER),
  print_list(Y).


selection(0,[],_):- fail./* convert menu select to value */
selection(1,[H|_],H):- !.
selection(N,[_|T],Item):-
  N1=N-1,
  selection(N1,T,Item).

update_class_members(C,M):-
  class_members(C,ML),
  retract(class_members(C,ML)),
  insert_sort([M|ML],SML),
  assertz(class_members(C,SML)),
  !.

update_class_members(C,M):-
  assertz(class_members(C,[M])),
  !.
```

Vita

The author was born in Abington, Pennsylvania, on February 11, 1957, and is the son of William H. Fenton and E. Charlotte Fenton. Upon graduating from Northeast High School, Philadelphia, Pennsylvania, in June of 1975, he attended Clarion State University (now Clarion University), Clarion, Pennsylvania. He graduated in May of 1979 with a Bachelors of Science in Business Administration. Upon graduation from Clarion he was employed at Air Products and Chemicals Inc., Allentown, Pennsylvania, where he was a commercial business systems programmer. In January of 1985 he took a Project Manager position with Systems and Computer Technology, Malvern, Pennsylvania, where he managed the development of a student information system for Colorado State University, University of California at San Diego, Michigan State University, and University of Southwest Louisiana. In March of 1987 he took a Database Analyst position with Sorbus, Frazer, Pennsylvania. In January of 1988 he was transferred to Bell Atlantic Enterprises, Princeton, New Jersey, where he administers database management systems to support application systems development within Bell Atlantic Enterprise subsidiaries. He currently holds the title of Senior Database Analyst.