The Design of a Small Business Database using the Semantic Database Model

by

Jac F. Morgan

B.S., Kansas State University, 1975
M.S., Kansas State University, 1978

A MASTER'S REPORT

Submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

Approved by:

[Signature]
Major Advisor
Table of Contents

List of Figures ........................................... 1

Introduction - Chapter 1 .............................. 1

The problem environment - Chapter 2 .................. 7

An overview of SDM - Chapter 3 ......................... 12

A description of the SDM - Chapter 4 ................. 22

SDM application - Chapter 5 ............................ 72

Converting an SDM model to a relational model - Chapter 6 .... 96

Conclusion - Chapter 7 .................................. 103

SDM DbDL syntax (BNF) - Appendix A ................. 110

SDM DbDL syntax (Warnier) - Appendix B .............. 112

List of classes and attributes after step 3 - Appendix C .... 117

Final SDM schema for insurance agency - Appendix D .... 122

Forms used by insurance agency - Appendix E .......... 148

References .............................................. 159
THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING BEING SKEWED DIFFERENTLY FROM THE TOP OF THE PAGE TO THE BOTTOM.

THIS IS AS RECEIVED FROM THE CUSTOMER.
ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE
List of figures appearing in this report

<table>
<thead>
<tr>
<th>Figure number and title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1 Example of a class description</td>
<td>15</td>
</tr>
<tr>
<td>3-2 Example of a subclass description</td>
<td>15</td>
</tr>
<tr>
<td>3-3 Example of interclass connection describing a subclass</td>
<td>17</td>
</tr>
<tr>
<td>3-4 Example of a subclass description</td>
<td>17</td>
</tr>
<tr>
<td>3-5 Example of inverse attribute interrelationship</td>
<td>18</td>
</tr>
<tr>
<td>3-6 Example of match attribute interrelationship</td>
<td>19</td>
</tr>
<tr>
<td>3-7 The target class of the match in figure 3-6</td>
<td>20</td>
</tr>
<tr>
<td>5-1 Initial entities and events</td>
<td>78</td>
</tr>
<tr>
<td>5-2 The subclass approach to entity relationships</td>
<td>80</td>
</tr>
<tr>
<td>5-3 The grouping approach to entity relationships</td>
<td>80</td>
</tr>
<tr>
<td>5-4 Initial assessment of entities and events</td>
<td>83</td>
</tr>
<tr>
<td>5-5 Entities and events described as SDM categories</td>
<td>84</td>
</tr>
<tr>
<td>5-6 A class description demonstrating two methods of matching when</td>
<td></td>
</tr>
<tr>
<td>the target is an attribute class</td>
<td>88</td>
</tr>
<tr>
<td>5-7 Attribute class, the target of the match</td>
<td>89</td>
</tr>
<tr>
<td>5-8 Base class, the target of the match</td>
<td>89</td>
</tr>
<tr>
<td>5-9 Diagram showing the 2 match strategies</td>
<td>90</td>
</tr>
<tr>
<td>5-10 Attribute class, the target of the match</td>
<td>89</td>
</tr>
<tr>
<td>5-11 Diagram showing matching on different attribute names</td>
<td>91</td>
</tr>
<tr>
<td>5-12 Base class, the target of the match</td>
<td>92</td>
</tr>
<tr>
<td>5-13 Attribute description requiring multiple value classes</td>
<td>92</td>
</tr>
<tr>
<td>5-14 Attribute description requiring multiple value classes</td>
<td>93</td>
</tr>
<tr>
<td>6-1 Classes and attributes remaining after eliminating derived attributes and name classes</td>
<td>97</td>
</tr>
<tr>
<td>6-2 Relational model of the insurance agency</td>
<td>100</td>
</tr>
<tr>
<td>7-1 Description using alternative match attribute interrelationships</td>
<td></td>
</tr>
<tr>
<td>descriptions</td>
<td>106</td>
</tr>
<tr>
<td>7-2 Proposed modifications to the match attribute interrelationships</td>
<td>106</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Database management systems were developed to bring the data resources of an enterprise into a centralized, manageable system. They were intended to bear the burden of the management of data, specifically to insure data compatibility; eliminate or reduce data duplication; create a programming environment free from considerations of the physical storage of data; centralize the management of data; and enhance programmer productivity.

The implementation of a database management system involves gathering all the data used by the enterprise and determining its use. Because a major objective of the DBMS implementation is to reduce data redundancy, thus enhancing data currency and consistency, each project's data must be integrated with data from all other projects. Even if the implementation is not that ambitious and only one section's data is to be incorporated into the DBMS, the scale of the project is diminished but not its complexity.

Database (DB) designers must understand the meaning of all the data elements and their relationships to all other data elements that will populate the DBMS. They must also keep an eye on the future and create a design that will facilitate the addition, deletion and modification of data within the DB. All of this requires an extraordinary amount of detailed knowledge of the data's organization.

Realizing the inordinate complexity and management nightmares that are associated with such an undertaking, researchers have proposed several database design methodologies.
A database design life cycle was proposed by Unger and Fisher (14) which identified six phases. They are as follows:

1. Predesign evaluation
2. Information modeling
3. Logical database design
4. Database management system selection
5. Cost/benefit analysis
6. Physical design and implementation

Predesign evaluation, the first phase, is accomplished by collecting information about the enterprise, the data it uses and how it is used. This will provide a perspective of what is needed and the expectations of the individuals who will be using the system. If after phase one it is decided that a database is the most feasible approach then phase two begins.

This report will be concerned with the second phase of the design life cycle, information modeling which is often referred to as conceptual modeling. Conceptual modeling involves using the information collected in phase one to create, on paper, a facsimile of the day to day operations of the enterprise, e.g., the data requirements and processing requirements of the users. The most common method for accomplishing this task is to view the data and processes of the enterprise as a collection of "things" that are relevant to the enterprise. An intuitive knowledge of these "things" is required to insure a successful database implementation. The concept of "things" seems so ambiguous, yet an understanding of them is of such importance that a few paragraphs concerning the nature of things is essential to the successful conceptual and logical modeling of a database.

Many authors have addressed the issue of "things", however, Hsu and Roussopoulos (5) have provided an extremely illuminating description which places things in perspective and which can be summarized in what follows:

Things in the real world can be concrete or abstract --- houses, people,
ships are concrete while time, dates, phone numbers and events are abstract. Intuitively we all recognize and understand what things are, but in order to model them we must break them into smaller "definable" parts. We thus say that "things" consist of entities, properties and relationships.

Entities

Entities and objects are often used synonymously when describing objects in an application environment. Hsu and Roussopoulos (5) put it quite well when they said,

"Entities are a state of mind ... Intuitively, an entity may be defined as something that is of interest in a given context and it may have properties that are of interest in the same context. A property may be thought of as something that characterizes an entity or set of entities."

The value of these properties can, at times, be used to uniquely identify an entity. It must be realized that the difference between entities and their properties depend on the context in which they are viewed. In this case context not only refers to a current state but also the perspectives of different individuals when viewing the same state. The latter is referred to as user views and will be discussed later. It is argued that an entity may exist on its own but a property can exist only if the entity it characterizes also exists. Once again we must emphasize the fact that the distinction is relative to the context in which they are used. For example, from Hsu (5)

"This car has red color.
Color red has wavelength X."

The entity color in the first sentence is a property of the object car. In the second sentence the entity wavelength is a property of the object color.
**Relationships**

Relationships describe the way in which two or more entities are connected to define an action, a state, a concept, an object, or a transaction. A list of entities and their properties by themselves would impart little meaning without some means of relating the entities one to another. A relationship may exist between different entity types, e.g., a person owns an insurance policy (a relationship between a person and an insurance policy); or a relationship may exist between the same entity types, e.g., George is Jim's client (a relationship between two people). A relationship may exist as a one to one association between two entities, e.g., John's wife is Sally; a one to many association, e.g., John's children are Sue, Mark and Tom; or a many to many association, e.g., John has many friends and each friend has many friends including John. Relationships may also have properties. For example, John owns four insurance policies the total value of which is $250,000. The four policies are not attributes which characterize John, and the sum of $250,000 does not characterize each policy, so it must characterize the association between John and his insurance policies. Because entities may be abstract things in the real world and they are characterized by properties, it can be argued that relationships may be viewed as abstract entities. Hsu and Roussopoulos defended this assertion in the following manner:

"In our view, associations are entities. Since no two people will agree on what a real world view is, entities are a state of mind. Our view is based on the observation that an association may be a property of another object. The entity of supply association between suppliers and parts has property, the quantity; viz., suppliers supply parts in quantities. The supply association is an object in this case. The entity of birth date association among month, day and year is a property of a person. We believe our view of regarding associations as entities is a pragmatic one, because our view is consistent with the human mental process of abstractions."
The Semantic Database Model (SDM) (2,3,4,10,11) is a conceptual modeling tool which was designed not only to specify entities and relationships but also to provide a means for formally adding the meaning of entities and relationships to the model. As with other conceptual models (1,6,7,12,13) the SDM views a database as a collection of entities and relationships between entities. It diverges from the other models in the manner in which these relationships are described.

One's view of a conceptual model may range from the data section of a COBOL program to Sowa's conceptual graphs (12). The former view pertains more to a physical implementation and the latter view provides a framework extensive enough for the development of artificial intelligence in the computer. The COBOL description of an application's environment is rejected as a conceptual model because it has been more or less agreed upon that a conceptual model should be free of logical as well as physical implementation considerations. The degree of sophistication of a modeling tool then becomes a function of the application to be modeled. Sowa measures the difference between artificial intelligence systems and database systems as a ratio of data descriptors to individual data items. In a database system one descriptor may describe thousands of records whereas in an artificial intelligence application there is so little repetition that each record may have its own descriptor. Sowa's graphs were developed to provide an exhaustive description of a unique application environment. There is no argument that his graphs would fulfill the qualitative requirement of a database conceptual model, however, the model would be voluminous and too complex for the normal user to understand.

Of the conceptual models introduced to date, the SDM was chosen for study because it seemed to provide a facility for incorporating meaning into a conceptual model without placing it beyond the comprehension of the people who
are to manage and use the database.

This report will deal with the creation of an SDM schema of an insurance agency. The agency has been the subject of a database class design project and is currently being used as a pedagogical tool for teaching a graduate database course. The information and knowledge acquired to date will be used to assist in the application of the SDM database definition language (DbDL) to this problem domain. The purpose of this endeavor is two fold. First, it is written in an attempt to provide a comprehensive guide to the use of the SDM as a conceptual modeling tool in sufficient detail that it may be used as a pedagogical tool, and second, to evaluate the strengths and weaknesses of the SDM in representing the semantics of the problem domain. As a teaching tool this report will at times appear somewhat redundant, a necessary aspect of instruction.

This chapter has discussed the need for a viable conceptual modeling tool. Chapter 2 will provide the reader with a description of the enterprise to be modeled. Chapter 3 will present an overview of the SDM in general terms without exposing the reader to specifics of the model's syntax. Chapter 4 will provide a detailed description of the SDM's DbDL and demonstrate its use with examples from the insurance agency schema. If needed, situations not present within the enterprise schema will be contrived in order to demonstrate specific points. Chapter 5 will provide steps to follow when applying the SDM to a problem domain and detail their use in the development of an SDM schema for the insurance agency. Chapter 6 will demonstrate a method for converting an SDM model into a relational model. Chapter 7 will critique the SDM in light of experiences gained by its use and also provide suggestions as to the direction of future study.
Chapter 2

The problem environment

This chapter will describe the insurance agency to be modeled. It will provide the information that would have been gathered by on-site visits to the agency. From the narrative presented here and an examination of the forms used by the agency which appear in Appendix E, the reader may begin to form a conceptual image of the enterprise and the "things" that will populate the conceptual schema.

The agency has one general agent, i.e., an individual who has primary responsibility to the various insurance and investment companies for whom the agency sells products. There are agents (alias subagents) who are responsible to the general agent, and an office manager who manages the office staff.

The agency sells products for three insurance companies, annuities for two companies, and IRA's for three companies at the present time. The number of companies for which products are sold increases occasionally and presently there is consideration to expand the types of products offered to include some of the limited partnership products available.

Each agent has to be licensed to sell a type of product and then under contract to a company to sell that product type. A commission accrues for each product and the company. The commissions are always a percentage of the client payment. The general agent receives an override commission on all products sold by the agency. This override commission is determined by the product and the company and is expressed as a percentage of the client payment.

When the agency was originally visited seven documents were gathered. These documents were studied to determine the type and volume of data that was to populate the database. Close examination of the forms will reveal much
duplicate information. It is the job of the database designer to identify the types and categories of data and with the assistance of the employees of the enterprise, determine what data should go into the database. The forms collected were as follows: a) client file card (p. 148); b) insurance application (p. 149); c) client asset sheet (p. 153); d) estate settlement worksheet (p. 154); e) cash flow statement (p. 155); f) pro forma cash flow statement (p. 156); g) statement of financial position (p. 157); and policy record (p. 158).

The client file cards form the easy access role for the agent to use on a daily basis. A facsimile of this document is needed by the agents when they have to respond to phone requests by the client and for information to meet most of the daily reminder events.

The life insurance form is filled out and sent to the home office when insurance is requested. A copy is kept in the clients folder. Very few pieces of data are needed from this form for continued interaction with the client. These include the type and amount of insurance and other information required on the policy record (p. 158).

The client asset sheet is a data collection document; the data is used by the agents to run a financial analysis for the client. This form is to be stored with the client file and updated each year. Totals for the four columns of information are to be calculated.

The estate settlement worksheet is created for each spouse as part of the financial analysis that is provided by the agency as a service to its clients. This form is filled out for both husband and wife if that is appropriate.

The cash flow statement is a device to aid clients in the determination of the amount of money they can expect in investments, etc. It and the pro forma cash flow statement are always filled out by the client.

The Statement of Financial Position is a form that the agent has the
client fill out preparatory to creating a financial analysis for the client.

The policy record sheet is an often referenced document on a client. It provides the agent with information from the policies the client has in force and provides information to assist the agent in preparing the annual review. The cash values change each year and must be updated by the secretaries prior to the client's annual review.

**Reports**

The daily contact report is a reminder system which currently is kept on filing cards. Its purpose is to remind the agents and secretaries of events that will occur, e.g., appointments and things which must be done that day. Such things may be reminders of annual reviews for clients, policy issuance follow-up, client courtesy cards (e.g., birthday cards), bills to be paid, etc. These events may be added to the daily card up to one year prior to the day they must be processed.

Most events added to the daily contact file may be generated automatically. Annual reviews should be scheduled at one year intervals from the date of the last scheduled review. Thus a reminder to schedule this appointment should be inserted one week before a year has elapsed since the last review. The reminder for the next year should be inserted one year shy one week from the last review. For courtesy cards a reminder should be inserted into the file 3 working days before the actual birthday.

Client follow-up for insurance clients can be scheduled automatically. This procedure begins when an APS form is received by the office manager from the home office of the insurance company. After 5 working days have elapsed, the secretary should be reminded to call the home office of the insurance company to see if the doctor's follow-up to the APS form has been received. If it has been received then no additional reminders are needed; if it has not, the secretary should immediately call the doctor's office. If that office says
the form has been sent, in 2 days the secretary should again be reminded to call the home office. Also, upon receipt of a second notice APS form the secretary should call the doctor's office in 4 working days. This procedure continues until the APS is received by the home office. All calls should be recorded in the client's record for the agent's information.

Reinstatement follow-up for insurance clients can also be automated. This process is initiated when a client fails to pay an insurance premium on time and subsequently payment is sent. The agent should be notified in two weeks to check with the home office to see if the forms and the money have been received.

When an agent sells insurance to an out-of-town client, the secretary should be reminded in two weeks to check on receipt of the application by the agency.

Replacement follow-up for insurance clients is still another procedure which may be done semi-automatically. Here the goal is to minimize the payment for the client. Two weeks after the replacement policy is sold the secretary should be reminded to contact the insurance home office to see if the policy has been issued; if so, the client is to be notified immediately to stop payment on the replacement contract.

Prospect letters are sent daily. The follow-up for these should take place 3 days later by the agent.

All daily reminder items are to be automatically scrolled to the next working day if they are not accomplished.

Other Output Reports Required

Quarterly and annual reports of the state of the business are required for management purposes. These reports simply list the income and the expenses of the business. These, at the present time, are backed-up by detailed listings monthly. An entry for an expense gives the date, description, amount
and the individual to whom the expense is charged. An entry for the income gives the date, amount and the individual who brought that money into the business. All commissions have been divided by the companies at this point. Only the incomes to the general agent are considered income to the agency.
Chapter 3

An overview of the SDM

This chapter will introduce the reader to the syntax of the SDM using a minimum amount of its database definition language (DbDL) and little detail. The intent is to ease the reader into a general understanding of the SDM and how it can be used before exposing him to a detailed description of its DbDL syntax (Chapter 4). While reading this chapter the reader should keep the following facts in mind.

1. An SDM schema is a conceptual representation of the application environment. Implementation consideration should be ignored. In fact, approaching the SDM with the databases ultimate implementation in mind will hamper your understanding of the use and potential of this conceptual tool.

2. A schema is constructed entirely of class descriptions.

3. A class represents an entity or collection of entities.

4. Attributes describe the properties of the class.

5. Classes are related by interclass connections, attribute interrelationships and interattribute derivations.

6. The value of an attribute is selected from a domain of values described by a value class, or it calculated (derived) using information within the database.

These aspects will be discussed in general terms in the remainder of this chapter.

The basic building block of the SDM is the class (4). A class is an entity or meaningful collection of entities. Entities represent the objects found in the application environment. They may be concrete objects such as
agents, secretaries, or policies sold by one company; events such as contact reminders or renewal notices; or higher-level entities such as all life insurance policies sold by one company or all employees of the enterprise. You may have entities which provide representations of the values of the entities that populate the database. The syntactic identifiers used to represent values are referred to as name classes. Name classes are always described as a subset of the set of all possible strings, integers or reals. Examples of names include: all possible calendar dates, all possible first names, all possible social security numbers, etc. All of these -- concrete objects, events, higher-level entities, names -- are described relative to the user's perspective.

A database designer may view a class as a logical grouping of entities. For example, one class describing the enterprise may represent its employees. The employee class might contain such descriptors as the employees name; personal information about the employee; information about the employee's spouse and children; the date the employee joined the company; something that would identify the employee's position within the enterprise, i.e., an agent, a subagent, an office worker, etc.; the employee's rate of pay; a ranking which implies the employee's seniority with respect to the other employees of the enterprise; and finally a count of the number of enterprise employees. These entities are grouped together because they constitute a reasonable description of an employee and thus form a logical grouping. The SDM representation of the employee class appears in figure 3-1. The class description in figure 3-1 probably seems ominous and confusing and rightly so, it's written in a language that most are not familiar with and which is the subject of the remainder of this report. It would be a good idea to take a few minutes to study figure 3-1 to get an impression of what a class description looks like. Don't be concerned if it appears foreign to you --- it should!
The organization and structure of the database is specified by an SDM schema which consists entirely of class descriptions. A class description contains 1) a class name including all synonyms, 2) an optional class description, 3) a description of any relationships in which the class may participate, and 4) a list of attributes. Attributes describe the class and derive their values from other classes (the attribute's value class) or from calculation performed on information contained within the database (derived attribute). Relationships between entities are both implicitly and explicitly described. To list an attribute's value class implicitly implies a relationship between the class defined by the attribute and the class providing the value for the attribute. The attribute Status_of in the class EMPLOYEE has as its value class EMPLOYEE_TYPES (figure 3-2). This says that the value of Status_of is selected from the domain of values described by the class EMPLOYEE_TYPES (the implicit relationship). (The SDM syntax requires that class names appear in full capital letters and attribute names appear with their first letter capitalized.) EMPLOYEE_TYPES, in turn, is defined as a subclass (interclass connection) of the built-in class STRINGS and is further restricted to values that are specified by the user. This condition is denoted by the "where specified" within the description of EMPLOYEE_TYPES. The class STRINGS is included within the SDM by its authors and is defined as the basic set of alphanumeric characters (1). The occurrence of several member attributes in the definition of EMPLOYEE implies that the entities PERSON_NAMES, PERSONAL_INFO, SPOUSE_INFO, DEPENDENT_INFO, etc., form a relationship with the entity EMPLOYEE and also a relationship with one another (they form the set of entities from whose domains the attributes defining EMPLOYEE acquire their values).

Explicit relationships are described by interclass connections, attribute
EMPLOYEE

description: all people who work for the enterprise.

member attributes:

  Employee_name
    value class: PERSON_NAMES

  Employee_personal_info
    value class: PERSONAL_INFO

  Employee_spouse_info
    value class: SPOUSE_INFO

  Employee_dependent_info
    value class: DEPENDENT_INFO
    multivalued

  DateJoined_company
    value class: DATES

  Status_of
    value class: EMPLOYEE_TYPES

  Pay
    value class: PAY

  Seniority
    value class: INTEGERS
    derivation: order by increasing DateJoined_company

class attributes:

  Number_of_employees
    value class: INTEGERS
    derivation: number of members in this class.

identifiers: Employee_name

Figure 3-1: An example of a class description using the SDM DbDL.

EMPLOYEE_TYPES

description: The types of employees at the agency, values may be
"agent", "subagent" and "office staff".

interclass connection: subclass of STRINGS where specified

Figure: 3-2: An example of a subclass description.
interrelationships and interattribute derivations. An example of the first is found in the definition of EMPLOYEE_TYPES. EMPLOYEE_TYPES is defined as a subset of the built-in class STRINGS. Once again the class STRINGS is a built-in class of the SDM and it represents the domain of all possible strings. Another example of an interclass connection is found in the definition of AGENT (figure 3-3). AGENT is a subset of EMPLOYEE where the value of the attribute Status_of = agent. By virtue of attribute inheritance from parent class to subclass the attributes of the class EMPLOYEE aid in the description of AGENT if the employee in question is an agent. If the employee in question is an office worker then those attributes describing EMPLOYEE aid in the description of OFFICE_STAFF (figure 3-4). A second type of interclass connection allows the grouping of classes into aggregates based on common values of one or more attributes, on enumerated groups or a user-controllable grouping.

Attribute interrelationships are provided by SDM to accommodate multiple views of the data. Binary associations can be established between entities of a class by defining each to be the inverse of the other. For example, the attribute Agent_of_record of the class CLIENT is specified as the inverse of attribute Client_of of the class AGENT_INFO (figure 3-5) Both then provide a perspective as to a clients agent and an agent's clients. Binary and higher degree association are accomplished by matching the value of an attribute with an attribute of an entity that also possesses data desired. The insurance agency conceptually requires a high degree of redundant data. This semantic redundancy is supported in SDM in a conceptual manner. SDM allows the database designer to reference data which already occurs within the database by matching. For example, the values of the attributes of the class CLIENT_FILE_CARD_POLICY_INFO (figure 3-6) are obtained from a particular instance of the class POLICY_RECORD (figure 3-7) by matching on the value of
AGENT

description: The owner and primary agent of the enterprise.
interclass connection: subclass of EMPLOYEE where Status_of="Agent"
member attributes:
  Agent_particulars
  value class: AGENT_INFO
  
  Agent_commission
  description: The agent receives a commission on all products sold by the agency. The rate varies with type and age of policy, the details of which are beyond the scope of this report.
  value class: DOLLARS

Figure 3-3: An example of an interclass connection which describes a class as a subclass of another class.

OFFICE_STAFF

description: The class that identifies the office staff and the office manager.
interclass connection: subclass of EMPLOYEE where Status_of = "office worker"
member attributes:
  Is_office_manager
  description: "yes" if the instance is the office manager, "no" otherwise.
  value class: BOOLEAN

Figure 3-4: Example of a subclass description.
CLIENT

description: Individuals who purchase products from the enterprise.
member attributes:

  Client_name
    value class: PERSON_NAMES
  Client_personal_info
    value class: PERSONAL_INFO
  Client_spouse_info
    value class: SPOUSE_INFO
  Client_dependent_info
    value class: DEPENDENT_INFO
    multivalued
  Client_business_info
    value class: CLIENT_BUSINESS_INFO
  Miscellaneous_information
    value class: MISC_INFO
  Annuities_owned
    value class: ANNUITIES_INFO
    multivalued
  IRA's_owned
    value class: IRA_INFO
    multivalued
  Agent_of_record
    value class: AGENT_INFO
    inverse: Client_of
  identifiers: Client_name

AGENT_INFO

description: Attributes that the agent and subagents have in common.
  Used as a value class for Agent_particulars of AGENT and
  Subagent_particulars of SUBAGENT.
member attributes:

  Clients_of
    value class: CLIENT
    inverse: Agent_of_record
    multivalued
    exhausts value class
    no overlap in values
  Licensed_with
    value class: COMPANIES
    multivalued
  Number_of_clients
    value class: INTEGERS
    derivation: number of unique members in Client_of

Figure 3-5: An example of two classes which utilizes the attribute
interrelationship, inverse.
CLIENT_FILE_CARD_POLICY_INFO

description: A summary of the policies a client owns.

member attributes:

Policy_number

description: client will have 1 instance of POLICY_RECORD for each policy owned. The policy numbers of the policies owned by each client can be derived from POLICY_RECORD. A match on the clients name will result in several policy numbers each of which should be used in turn to provide multiple instances of Client_policy_info, an attribute of CLIENT_FILE_CARD whose value class is CLIENT_FILE_CARD_POLICY_INFO.

value class: POLICY_NUMBERS
match: Policy_number of POLICY_RECORD on Client_name (of CLIENT_FILE_CARD)

Plan

value class: POLICY_TYPES
match: Type of POLICY_RECORD on Policy_number

Company

value class: COMPANY_NAMES
match: Company of POLICY_RECORD on Policy_number

Mode_of_premium_payment

value class: MODE_OF_PAYMENT

Premium_amount

value class: DOLLARS
match: Premium_amount of POLICY_RECORD on Policy_number

Date_premium_due

value class: DATES
match: Date_premium_due of POLICY_RECORD on Policy_number

Face_amount

value class: DOLLARS
match: Face_amount of POLICY_RECORD on Policy_number

Date_of_issue

value class: DATES
match: Date_of_issue of POLICY_RECORD on Policy_number

Age_of_issue

value class: AGE
match: Age_of_issue of POLICY_RECORD on Policy_number

Primary_beneficiary

value class: PERSON_NAMES
match: Primary_beneficiary of POLICY_RECORD on Policy_number

WP_rider

value class: BOOLEAN
match: WP_rider of POLICY_RECORD on Policy_number

Dividend_option

value class: BOOLEAN
match: Dividend_option of POLICY_RECORD on Policy_number

Figure 3-6: An example of a class which derives attribute values from attributes of another class (POLICY_RECORD, figure 3-7).
POLICY_RECORD

description: a collection of specifications about insurance policies owned by each client. The POLICY_RECORD class will act as the master record of each policy sold.

member attributes:
Client_name
  value class: PERSON_NAMES
Policy_number
  value class: POLICY_NUMBERS
Type
  value class: POLICY_TYPES
Company
  value class: COMPANY_NAMES
Face_amount
  value class: DOLLARS
Date_of_issue
  value class: DATES
Age_of_issue
  value class: AGE
Primary_beneficiary
  value class: PERSON_NAMES
WP_rider
  value class: BOOLEAN
Dividend_option
  value class: BOOLEAN
Gross_yearly_premium
  value class: DOLLARS
Add_rider
  value class: BOOLEAN
Cpd_rider
  value class: BOOLEAN
Apl_rider
  value class: BOOLEAN
Loans
  value class: LOAN_INFO
Cash_value_record
  value class: CASH_VALUE_SCHEDULE
Contingent_beneficiary
  value class: PERSON_NAMES
Total_insuranceOwned
  value class: DOLLARS
derivation: sum of Face_amount
Total_gross_yearly_premium
  value class: DOLLARS
derivation: sum of Gross_yearly_premium

identifiers: Client_name + Policy_number

Figure 3-7: A class from which attribute values are derived by other classes.
Policy_number.

Derived attributes provide a method for describing a value that is algorithmically derived from the information in the database. Derived values are common within application programs but are not normally included within a conceptual model. The addition of derived attributes within the SDM adds another layer of meaning to the relationships and use of the data. The attribute Seniority within the class EMPLOYEE describes a method for deriving the seniority status of an employee based on the date the employee joined the company.

The intent of this section was to prepare the reader for a more formal description of the SDM DbDL presented in Chapter 4. The reader is encouraged to review the six points mentioned at the beginning of this chapter and to study the figures presented. Some of them will appear again in Chapter 4.
Chapter 4

A description of the Semantic Database Model

The organization and structure of the database is specified by an SDM schema, which identifies the classes in the database and explicitly defines the relationships between them. The authors of the SDM have provided a detailed syntax of the Database Definition Language (DbDL) used by the SDM. The DbDL is presented in Appendix A in Backus-Naur form and in Appendix B in Warnier diagrams (15). This chapter will step through the DbDL, presented in Appendix A, explaining the various syntactic forms and providing examples in an effort to familiarize the user with the SDM and its application. This chapter is organized differently from the other chapters in that it is intended to be used as a student handout. For this reason the chapter was written in the form of a manual. The major topics covered will appear double underlined and in bold type at the top left of each page that deals with its description. Its syntax will then be presented followed by a description of the syntax and how it is used. Finally examples demonstrating the topic will be presented. Most may be found in Appendix D. This general format will be followed throughout this chapter, however, the presentation format will be altered from time to time to accommodate specific situations. A table of contents of this student manual appears on the following page.
Table of Contents

25 SDM formalisms
26 Class
28 Class name and class description
29 Attributes
33 Base class feature
33 Identifier
33 Attribute name
35 Interclass connection
36 Subclass
36 Attribute predicate
37 Specified
38 Set-operator-defined subclasses
38 Intersect (is in CLASS_NAME and is in CLASS_NAME)
39 Union (is in CLASS_NAME or is in CLASS_NAME)
40 Difference (is not in CLASS_NAME)
40 Is a value of ATTRIBUTE_NAME of CLASS_NAME
41 Format is FORMAT
42 Attribute predicate
42 Simple predicate
42 MAPPING SCALAR_COMPARATOR [CONSTANT, MAPPING],
42 MAPPING SET_COMPARATOR [CONSTANT; CLASS_NAME; MAPPING]
46 Grouping
46 Expression_defined grouping class (GROUPING of CLASS_NAME on common value of <ATTRIBUTE_NAME> (groups defined as classes are <CLASS_NAME>))
48 Enumerated grouping class (GROUPING of CLASS_NAME consists of classes <CLASS_NAME>)
48 User-controllable grouping class (GROUPING of CLASS_NAME as specified)
50 MEMBER_ATTRIBUTES
50 ATTRIBUTE_NAME and ATTRIBUTE_DESCRIPTION
51 Value class: CLASS_NAME
51 Inverse - Match
Attribute derivations
51 Single valued; multivalued {with size between CONSTANT and CONSTANT}
53 May not be null
53 Not changeable
53 Exhausts value class
54 No overlap in values
55 INVERSE
57 MATCH
59 ATTRIBUTE_DERIVATIONS
60 INTERATTRIBUTE_DERIVATIONS
60 Same as MAPPING
61 Subvalue of MAPPING where is in [CLASS_NAME; ATTRIBUTE_PREDICATE]
61 Where is in MAPPING and is in MAPPING; is in MAPPING or is in MAPPING; is in MAPPING and is not in MAPPING
62 = MAPPING_EXPRESSION
63 [Maximum; minimum; average; sum] of MAPPING
64 Number of {unique} members in MAPPING
MEMBER-SPECIFIED_DERIVATIONS
  Order by [increasing; decreasing] <MAPPING> {within <MAPPING>}
  If in CLASS_NAME
  [up to CONSTANT; all] levels of values of ATTRIBUTE_NAME
  Contents
CLASS_Attributes
CLASS-SPECIFIED_DERIVATIONS
Before proceeding with a description of the SDM, the reader must be aware of the items and terms used within the SDM syntax. A list of the formalisms and symbols used within the DbDL has been offered by Hammer and McLeod (4) and appear in the following:

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 "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.
CLASS

Syntax

CLASS <-
  <CLASS_NAME>
  [description: CLASS_DESCRIPTION]
  [[BASE_CLASS_FEATURES;INTERCLASS_CONNECTION]]
  [MEMBER_ATTRIBUTE]
  [CLASS_ATTRIBUTE]

Description

The syntax diagram for a Class is read to mean that: 1) a class is constructed of one or more identifiers called CLASS_NAME (synonyms are defined by including more than one CLASS_NAME); 2) a class may have an optional description of category CLASS_DESCRIPTION; 3) a class may have an optional BASE_CLASS_FEATURE category or an INTERCLASS_CONNECTION category but not both; 4) a class may have MEMBER_ATTRIBUTES; and 5) a class may have CLASS_ATTRIBUTES.

Examples

EMPLOYEE
description: all people who work for the enterprise.
member attributes:
  Employee_name
    value class: PERSON_NAMES
  Employee_personal_info
    value class: PERSONAL_INFO
  Employee_spouse_info
    value class: SPOUSE_INFO
  Employee_dependent_info
    value class: DEPENDENT_INFO
    multivalued
  Date_joined_company
    value class: DATES
  Status_of
    value class: EMPLOYEE_TYPES
  Pay
    value class: PAY
  Seniority
    value class: INTEGERS
derivation: order by increasing Date_joined_company
class attributes:
  Number_of_employees
    value class: INTEGERS
derivation: number of members in this class.
  identifiers: Employee_name
CLASS

OFFICE_STAFF

description: The class that identifies the office staff and the office manager.
interclass connection: subclass of EMPLOYEE where Status_of="office worker"
member attributes:
  is_office_manager
    description: "yes" if the instance is the office manager "no" otherwise.
    value class: BOOLEAN
CLASS_NAME and CLASS_DESCRIPTION

Syntax

CLASS_NAMES <- *string of capitals including special characters.*

CLASS_DESCRIPTION <- *STRING*

Description

The CLASS_NAME definition thus says that a Class must have a class name made up of uppercase letters and special characters; the designer may also include multiple names for the same class. This then provides a method for dealing with synonyms directly within the conceptual model. EMPLOYEE and OFFICE_STAFF (see example under CLASS) are examples of class names. As the DbDL specifies, the names appear in capital letters and include special characters. The underscore character is used extensively in an SDM schema.

A class description should give a brief description of the class, the entities making up the class (if appropriate) and the role they have in the enterprise. The class description provides a means and a place for internal documentation within the semantic model.

Example

ANNUAL_REVIEW_SCHEDULE

description: An annual review is conducted for each client once a year. The next review is scheduled 1 year from the last review. A notice is posted in the event schedule one week prior to the review date. Some of the attribute values may be derived from attributes of CLIENT (match on client name).
In order to discuss BASE_CLASS_FEATURES and INTERCLASS_CONNECTIONS you must understand the nature of attributes and the role they play in the SDM. The next few paragraphs are intended to provide you with a basic understanding of attributes.

Classes have distinctive features called attributes that set them apart from other classes. The term attribute in its simplest form implies an association between a value from a domain of possible values to an entity. The SDM has expanded the meaning of an attribute value to include the class, thus, an attribute value is either a class in the database, a collection of logically related classes, or is derived. The domain of values from which an attribute's value may be selected is called its value class. Any class within the SDM schema may be referenced as the value class of an attribute. For the insurance agency, EMPLOYEE is a class with several attributes, one of which is Employee_name. Employee_name has a value class of PERSON_NAMES which associates a value from the domain of PERSON_NAMES to individuals within the class EMPLOYEE. However, PERSON_NAMES is a class with the attributes Title, First, Middle and Last. These attributes in turn associate values from the domain of titles, e.g., Mr., Ms., Dr., etc., to the attribute Title, and values from the domain of first names, middle names and last names to their respective attribute names. Thus it is natural to have a class as the value of an attribute.

These domains are described in more detail by the value classes from which the attribute's value is drawn. This example introduces the concept of a name class. The values populating a database represents actual values encountered in the application environment. These values are denoted by strings of symbols taken from the set of alphanumerical characters. A name
class is described as a subset of the built-in SDM class STRINGS. For convenience the name classes NUMBERS, INTEGERS, REALS and YES/NO (Boolean) are also recognized as built-in SDM classes. Name classes can be described in the following manners:

1. The class can be described as the intersect, union or difference of two other name classes.

2. The class can be described as a subset of some other name class with the predicate "where specified".

3. Constraints on the acceptable data values a class can have are specified by the predicate "format is FORMAT".

4. If constraints as in 1, 2 and 3 are not placed on the name class then all strings are valid values.

SDM recognizes two types of attributes, member attributes and class attributes. A member attribute describes a property of each member of a class and thus has a value for each member. Each member of EMPLOYEE has the attributes Name, Employee_personal_info, Employee_spouse_info, Employee_dependent_info, Date_joined_company, Status_of, Pay, Seniority and Number_of_employees. Number_of_employees, a class attribute, does not pertain to each member of the class, rather it describes a property of the class EMPLOYEE as a whole and has only one value for the class, i.e., the number of employees employed by the agency.
ATTRIBUTES

Examples

EMPLOYEE

description: all people who work for the enterprise.

member attributes:

  Employee_name
    value class: PERSON_NAMES
  Employee_personal_info
    value class: PERSONAL_INFO
  Employee_spouse_info
    value class: SPOUSE_INFO
  Employee_dependent_info
    value class: DEPENDENT_INFO
    multivalued
  Date_joined_company
    value class: DATES
  Status_of
    value class: EMPLOYEE_TYPES
  Pay
    value class: PAY
  Seniority
    value class: INTEGERS
    derivation: order by increasing Date_joined_company

class attributes:

  Number_of_employees
    value class: INTEGERS
    derivation: number of members in this class

identifiers: Employee_name

PERSON_NAMES

description: The form a person's name may take.

member attributes:

  Title
    value class: TITLES
  First
    value class: FIRST_NAMES
  Middle
    value class: MIDDLE_NAMES
  Last
    value class: LAST_NAMES

TITLES

description: The title a person may have preceding his name
  e.g., Dr., Mr., Ms., Mrs.

interclass connection: subclass of STRINGS where specified

FIRST_NAMES

description: The first name of a person.

interclass connection: subclass of STRINGS where specified
ATTRIBUTES
========

MIDDLE_NAMES
   description: The middle name identifies a person may have. It may be one or more initials or one or more names.
   interclass connection: subclass of STRINGS where specified

LAST_NAMES
   description: The last name of a person, it may be one or more names or hyphenated.
   interclass connection: subclass of STRINGS where specified
BASE_CLASS_FEATURE

Syntax

BASE_CLASS_FEATURE <- ([duplicates allowed;duplicates not allowed])
<<IDENTIFIERS>>

IDENTIFIER <- [ATTRIBUTE_NAME;ATTRIBUTE_NAME+IDENTIFIER]

ATTRIBUTE_NAME <- [string of lowercase letters beginning with a capital
and may include special characters.*

Description

SDM recognizes a class as being either a base class or a nonbase class
(interclass connection, discussed later). A base class is considered a
primitive entity in that it cannot be defined in terms of one or more
classes, although it may be made up of one or more entities. EMPLOYEE is a
base class. The database designer can specify that a base class may contain
duplicate instances of its members. The default, by definition (4) is
"duplicates allowed". If it is required that all of the member attributes
taken together comprise a unique identifier then "duplicates not allowed"
must be specified.

Although the DbDL syntax implies an order to the components of a class
description, the base class features are placed at the bottom of the
examples. This is done to be consistent with the examples provided by the
authors of the SDM (4). Each member of a base class is identified by its
identifier. An identifier is constructed of an attribute name or an
attribute name plus ("+" another identifier. The latter method for
describing an identifier is used in the class CLIENT_COURTESY_CARD.
BASE_CLASS_FEATURE
====================

Example

CLIENT_COURTESY_CARD
  description: Courtesy cards are send to clients on special occasions, e.g., birthdays.
  member attributes:
    Name_of_agent
      value class: PERSON_NAMES
    Client_name
      value class: PERSON_NAMES
    Client_address
      value class: ADDRESSES
      match: Client_personal_info.Address of CLIENT on Client_name
    Client_birthday
      value class: DATES
      match: Client_personal_info.Date_of_birth of PERSONAL on Client_name
    Date_to_send_card
      Value class: DATES
      derivation: Client_birthday - 3 working days
  identifiers: Client_name + Date_to_send_card
INTERCLASS_CONNECTION
========================

Syntax

INTERCLASS_CONNECTION <- [SUBCLASS;GROUPING_CLASS]

Description

A nonbase class does not enjoy an independent existence but must be defined in terms of one or more classes. The syntactic argument used to define a nonbase class is referred to as an interclass connection.

Classes may be partitioned into subclasses inheriting attributes from their parent class. The subclass must be of type nonbase because it is defined in terms of its parent class. OFFICE_STAFF is an example of a subclass and it also offers a very good demonstration of attribute inheritance. The subclass OFFICE_STAFF has only one attribute description which is used to identify the office manager. It inherits all of the attributes from its parent class EMPLOYEE (see Examples of ATTRIBUTES).

Classes may also be aggregated into groups based on one or more common attribute values. They differ from a class in that the grouping is not necessarily permanent; in this case the grouping is based upon the values of attributes. The group class provides a convenient interclass connection for aggregates with changing member classes.

Example

OFFICE_STAFF
   description: The class that identifies the office staff and the office manager.
   interclass connection: subclass of EMPLOYEE where
      Status_of="office worker"
   member attributes:
      Is_office_manager
         description: "yes" if the instance is the office manager
                      "no" otherwise.
         value class: BOOLEAN
SUBCLASS
========

Syntax

SUBCLASS <- subclass of CLASS_NAME where SUBCLASS_PREDICATE

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]

Description

A subclass contains some but not all of the members of the parent class. The factor that determines the entities that belong to the subclass is the subclass predicate. An entity can thus be a member of any subclass for which the subclass predicate is satisfied. This is one of the SDM features which provides for multiple views of the same data. A subagent who specializes in annuities may be interested in only those clients who own annuities. An appropriate subclass of clients from his viewpoint would be one containing only those clients. Another subagent whose specialty is IRA's may choose a viewpoint which includes only those clients who own IRA's. This subclass may include some or all of those included in the annuity subclass. Thus, different perspectives on the same data are provided.

ATTRIBUTE_PREDICATE

Description

The discussion of attribute_predicates (attribute defined subclasses) will require additional syntax diagrams and thus will occur as a major topic.
SUBCLASS
======

Specified

Description

The term "specified" defines a user-controllable subclass. Users of the database manually specify the entities which belong to this subclass. If a group of clients was selected for some type of preferential treatment (the agent's in-laws perhaps) a subclass of CLIENT called SPECIAL_CLIENTS could be defined that would contain only the members of CLIENT that were manually placed into it.

Examples

CLIENT
definition: individuals who purchase products from the enterprise.
member attributes:
    Client_name
        value class: PERSON_NAMES
    Client_personal_info
        value class: PERSONAL_INFO
    Client_spouse_info
        value class: SPOUSE_INFO
    Client_dependent_info
        value class: DEPENDENT_INFO
        multivalued
    Client_business_info
        value class: CLIENT_BUSINESS_INFO
    Miscellaneous_information
        value class: MISC_INFO
    Annuities_owned
        value class: ANNUITIES_INFO
        multivalued
    IRA's_owned
        value class: IRA_INFO
        multivalued
    Agent_of_record
        value class: AGENT_INFO
        inverse: Client_of
    identifiers: Client_name

SPECIAL_CLIENTS
description: Those clients selected to receive preferential treatment.
interclass connection: subclass of CLIENT where specified
Set-operator-defined subclasses

Description

The next three subclass types to be discussed, are referred to as set-operator-defined subclasses. Membership into subclasses defined using these predicates is based upon the attribute's membership within two other classes. An intersect relationship (is in CLASS_NAME and is in CLASS_NAME) specifies that the members of the subclass are just those members that belong to Class 1 \((C_1)\) and Class 2 \((C_2)\). Set union (is in CLASS_NAME or is in CLASS_NAME) requires that the members of the subclass be members of \(C_1\) or \(C_2\). Set difference (is not CLASS_NAME) limits the members of a subclass to those members of \(C\) but not in \(C_1\). To insure that \(C_1\) and \(C_2\) are of the same data type, \(C_1\) and \(C_2\) must both be subclasses of the same parent class either directly or through a series of subclass relationships.

Examples of the set_operator_defined subclasses were not readily available in the insurance agency SDM schema, because the schema is to reflect the actual working environment of the agency. For demonstration purposes artificial classes will be created when specific examples are needed to demonstrate a concept.

**Intersect**

Description

The schema might contain a subclass of CLIENT which contains only those clients who are 55 years of age or older. This type of subclass is called an attribute defined subclass and will be
discussed later. Its description is necessary to demonstrate the intersect relationship. If the database designer then wished to define a subclass which contained only those clients who were preferred clients and also over 55 years of age, the class Special_Clients_over_55 would be defined. This class would then include the members who belong to both classes.

Examples

SENIOR_CLIENTS
  description: Those clients 55 years of age or older.
  interclass connection: subclass of CLIENT where AGE=>55.

SPECIAL_CLIENTS_OVER_55
  description: Special clients who are 55 years of age or older.
  interclass connection: subclass of CLIENT where is in SPECIAL_CLIENTS and is in SENIOR_CLIENTS

Union

Description

If the database designer desired a subclass of all clients who are preferred clients and all those who are 55 years of age or older the class SPECIAL_CLIENTS_OR_SENIOR_CLIENTS would be defined. This subclass would contain the members of both classes.

Example

SPECIAL_CLIENTS_OR_SENIOR_CLIENTS
  description: All special clients and all clients 55 years of age and older.
  interclass connections: subclass of CLIENT where is in SPECIAL_CLIENTS or is in SENIOR_CLIENTS

39
**Difference**

**Description**

The agency's clients who are not yet 55 years old can be collected into their own subclass. YOUNGER_CLIENTS would contain those members of the class CLIENT that are not members of the subclass SENIOR_CLIENTS.

**Example**

YOUNGER_CLIENTS
  description: Those clients who are not yet 55 years old.
  interclass connection: subclass of CLIENT where in not in SENIOR_CLIENTS.

... is a value of ATTRIBUTE_NAME of CLASS_NAME ...

**Description**

This is a predicate that identifies the members of the subclass as being those that are already values of an attribute defining another class. This is referred to as an existence subclass. For example, a subclass of CLIENT which would contain those clients who are also employees (EMPLOYEE_PATRONS) is described as an existence subclass.

**Example**

EMPLOYEE_PATRONS
  description: Those employees who are also clients of the agency.
  interclass connection: subclass of CLIENTS where is a value of Employee_name of EMPLOYEE.
SUBCLASS
=========

**Format is FORMAT**

Syntax

```plaintext
FORMAT <- "a name class definition pattern"
```

Description

This is a clause that allows the database designer to explicitly specify the format of values which the subclass may contain (9). SOCIAL_SECURITY_NUMBER is described as a subclass of STRINGS where format is a 3 digit number between 000 and 999 followed by a hyphen then a 2 digit number between 00 and 99 followed by a hyphen then a 4 digit number between 0000 and 9999. Format is used primarily in the description of name classes.

Example

```plaintext
SOCIAL_SECURITY_NUMBER
  interclass connection: subclass of STRINGS where format is
    000<=number<=999
    "-"
    00<=number<=99
    "-"
    0000<=number<=9999
```
ATTRIBUTE_PREDICATE

Syntax

ATTRIBUTE_PREDICATE <-
    [SIMPLE_PREDICATE; ATTRIBUTE_PREDICATE; not ATTRIBUTE_PREDICATE;
     ATTRIBUTE_PREDICATE and ATTRIBUTE_PREDICATE;
     ATTRIBUTE_PREDICATE or ATTRIBUTE_PREDICATE]

SIMPLE_PREDICATE <- [MAPPING SCALAR_COMPARATOR [CONSTANT; MAPPING];
                     MAPPING SET_COMPARATOR [CONSTANT; CLASS_NAME: MAPPING]]

MAPPING <- [ATTRIBUTE_NAME; MAPPING.ATTRIBUTE_NAME]

SCALAR_COMPARATOR <- [EQUAL_COMPARATOR; >; >=; <; <= ]

EQUAL_COMPARATOR <- [ =, <> ]

CONSTANT <- *a string of number constants*

Set_Comparator <- [is {properly} contained in; {properly} contains]

Description

The attribute predicate is used to delimit the members of a subclass by
specifying the assertions which must be true for each member of the
subclass. The attribute predicate thus is defined as a simple predicate or
combination of simple predicates joined by one of the following Boolean
operators: "not", "and" and "or".

Mapping is a means of referring to specific values of an attribute.
In the examples both attribute names and mappings are used to reference
specific values. Example 1 demonstrates the use of the first option of the
mapping syntax (Attribute_name). The value being referenced is a specific
value of the attribute Status_of. A mapping may also represent a
concatenation of attribute names in order to allow direct reference to the
value of an attribute. In example 2 the first attribute being referenced is
Client_name Last. A period is used to separate the attribute names which
comprise the mapping. Mappings occurring within the narrative will be
elevated to the middle of the printed line to distinguish it from the
period ending the sentence. The attribute Last is found in the class PERSON_NAMES (example 6). PERSON_NAMES is the value class of the attribute Client_name of the class CLIENT (example 7), thus the notation Client_name.Last. The second value being referenced (Client_spouse_info.Name.Last) is the attribute Last of the class PERSON_NAMES which is a value class of the attribute Name of the class SPOUSE_INFO (example 8). SPOUSE_NAME is the value class of Clients_spouse_info of the class CLIENT.

By applying a simple predicate or a combination of simple predicates we are able to select from the members of a class, specific members that form a meaningful subclass of the original parent class. More complex stipulations can be formed by combinations of simple predicates joined by one or a combination of the boolean operators "and", "or" and "not".

Examples

1) Mapping scalar_comparator constant;

AGENT
    interclass connection: subclass of EMPLOYEE where
    Status_of = "agent"

2) Mapping scalar_comparator mapping;

CLIENT_WHOSE_SPOUSE_HAS_A_DIFFERENT_NAME
    interclass connection: subclass of CLIENT where
    Client_name.Last <> Client_spouse_info.Name.Last

3) Mapping set_comparator Constant;

INSURANCE_COMPANIES
    interclass connection: subclass of COMPANIES where
    Company_type contains "insurance"

4) Mapping Set_Comparator Class_Name;

AGENT'S_SENIOR_CLIENTS
    interclass connection: subclass of CLIENT where Client_of
    is in SENIOR_CLIENTS
5) Mapping Set_Comparator Mapping;

AGENT'S_PREFERRED_CLIENTS_OVER_55
interclass connection: subclass of CLIENT where Preferred_client
is in Client_over_55

6) PERSON_NAMES
description: The form a person's name may take
member attributes:
   Title
      value class: TITLES
   First
      value class: FIRST_NAMES
   Middle
      value class: MIDDLE_NAMES
   Last
      value class: LAST_NAMES

7) CLIENT
definition: individuals who purchase products from the enterprise.
member attributes:
   Client_name
      value class: PERSON_NAMES
   Client_personal_info
      value class: PERSONAL_INFO
   Client_spouse_info
      value class: SPOUSE_INFO
   Client_dependent_info
      value class: DEPENDENT_INFO
      multivalued
   Client_business_info
      value class: CLIENT_BUSINESS_INFO
   Miscellaneous_information
      value class: MISC_INFO
   Annuities_owned
      value class: ANNUITIES_INFO
      multivalued
   IRA's_owned
      value class: IRA_INFO
      multivalued
   Agent_of_record
      value class: AGENT_INFO
      inverse: Client_of
   identifiers: Client_name
8) SPouse_INFO
   description: That information relevant to a client's or
               employee's spouse.
   member attributes:
   Name
     value class: PERSON_NAMES
   Home_address
     value class: ADDRESSES
   Phone_number
     value class: PHONE_NUMBERS
   Date_of_birth
     value class: DATES
GROUPING
======

Syntax

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

Description

The second type of interclass connection used to describe a nonbase class is the grouping interclass connection. Grouping provides a means for describing higher level, abstract entities which are constructed out of basic entities.

Expression-defined grouping class

The first predicate defines a grouping class made up of members having a common value for one or more designated member attributes. For example, a class TYPES_OF_COMPANIES can be defined as a grouping class of COMPANIES based on common values of Company_type. Company_type thus represents a collection of types of companies whose instances are the members of the various types of companies represented by the agency. The contents of TYPES_OF_COMPANIES represent subsets of the class COMPANIES and thus correspond to a collection of attribute defined subclass definitions. For example, the subclass TYPES_OF_COMPANIES represents the grouping of subclasses which may be defined as Company_type = "insurance", Company_type = "IRA", etc. Because some of these subclasses may already be defined as subclasses within the SDM schema, it is recommended that these subclasses be referenced within the grouping class definition by naming the subclasses defined within the schema. INSURANCE_COMPANIES is described
as a subclass of COMPANIES where Company_type = "insurance".

This type of grouping is intended to focus attention on the shared properties of the entities that are its contents, rather than to the collection of entities itself. Although the name would infer an aggregation of classes, a grouping class still represents an abstract subclass of some underlying parent class.

Examples

TYPES_OF_COMPANIES
  description: Types of Companies represented by the agency.
  interclass connection: grouping of COMPANIES on common value of Company_type groups defined as classes are INSURANCE_COMPANIES, ANNUITY_COMPANIES :

INSURANCE_COMPANIES
  description: All insurance companies represented by the agency.
  interclass connection: subclass of COMPANIES where Company_type = "Insurance" :

ANNUITY_COMPANIES
  description: Companies which offer annuities through the agency.
  interclass connection: subclass of COMPANIES where Company_type = "Annuities" :

COMPANIES
  description: the companies whose products are sold by the enterprise.
  member attributes:
    Company_type
      value class: COMPANY_TYPES
    Company_name
      value class: COMPANY_NAMES
    Company_address
      value class: ADDRESSES
    Company_phone
      value class: PHONE_NUMBERS
    Products_offered
      value class: PRODUCT_TYPE
  identifiers: Company_name
ENUMERATED GROUPING CLASSES

Description

The second grouping predicate allows the database designer to specify the classes that will belong to the grouping class. However, the specific classes must all have been previously defined as a subclass of some underlying parent class. The class SPECIAL_CLIENT_GROUPS is described in such a manner. This grouping class description is useful when there is no appropriate attribute value which can be used to distinguish the various groups.

Example

SPECIAL_CLIENT_GROUPS
description: Those clients that are special to the agency at present are special clients and are those over 55 years of age.
interclass connection: grouping of CLIENTS consisting of classes SPECIAL_CLIENTS and SENIOR_CLIENTS.
::
::

USER-CONTROLLABLE GROUPING CLASSES

This last grouping predicate consists of a collection of user-controllable subclasses of some underlying parent class. It is a vehicle to use when the database designer wishes to define a class of clients considered "special cases", i.e., individual clients who require special attention which is not based on some common attribute value. The class would not contain groups of clients as in the first two cases. Each member of this class would be an individual client. Such a class would thus represent an aggregate of individual members which were chosen by the users.
GROUPING

Example
SPECIAL_ATTENTION

description: Those clients requiring special attention.
interclass connection: grouping of CLIENTS as specified

49
MEMBER_ATTRIBUTES

Syntax

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}

Description

SDM recognizes two types of attributes, member attributes and class
attributes. Member attributes describe properties belonging to each member
of the class and class attributes describe properties about the class in
general. The syntactic clauses shared by the two attribute types (attribute
name, attribute description, single valued-multivalued, may not be null,
not changeable) will be addressed during the discussion of member
attributes.

ATTRIBUTE_NAME and ATTRIBUTE_DESCRIPTION

Syntax

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

ATTRIBUTE_DESCRIPTION <- * string *

Description

For the purpose of standardization, attribute names will begin with
a capital letter with the remainder lowercase and may contain special
characters. An attribute name must be unique with respect to all
attribute names used in the class; it cannot have been used in its parent class if the class being defined is a subclass, and it cannot be used in any subclasses that may be defined from it. Synonyms are permitted. An attribute description may also be included and is encouraged to aid in the documentation of the schema.

Example

Spouse_amount_insurance

description: If it is assumed that if the spouse is also a client then a record of total insurance owned should also be present for the spouse. This value is derivable by matching Spouse_name to Client_name in POLICY_RECORD.

value class: DOLLARS

Value class: CLASS_NAME

Description

The value class of an attribute is the class which describes the domain of values an attribute may have. The attribute described in the above example has a value class of DOLLARS.

Inverse-Match

Inverse and match will be discussed as separate major topics.

Attribute derivations

Attribute derivations will be discussed as a major topic.

Single valued: multivalued (with size between CONSTANT and CONSTANT)

Description

The Agent_of_record attribute in the class CLIENT is an example of an attribute that may have only one value. Each agent sells to only his
own clients, they do not share clients. The Annuities_owned
attribute in the class CLIENT is an example of a multivalued attribute.
This attribute is a list of information pertaining to all annuities
sold to the client by the agent. If we think of a multivalued attribute
as being a subset of a value class, the number of values this set may
contain can be controlled by specifying the set size ("with size
between CONSTANT and CONSTANT"). For example, if for some reason we
wished to limit the number of clients an agent may have, we could
specify it in the manner displayed in a modified version of the class
AGENT_INFO.

Examples

CLIENT

definition: individuals who purchase products from the
enterprise.

member attributes:

Client_name
    value class: PERSON_NAME
Client.Personal_info
    value class: PERSONAL_INFO
Client_spouse_info
    value class: SPOUSE_INFO
Client_dependent_info
    value class: DEPENDENT_INFO
    multivalued
Client_business_info
    value class: CLIENT_BUSINESS_INFO
Miscellaneous_information
    value class: MISC_INFO
Annuities_owned
    value class: ANNUITIES_INFO
    multivalued
IRA's_owned
    value class: IRA_INFO
    multivalued
Agent_of_record
    value class: AGENT_INFO
    inverse: Client_of

identifiers: Client_name
AGENT_INFO

: member attributes:
  Clients_of
    value class: CLIENT
    multivalued with size between 1 and 150

: :

May not be null

Description

Attributes may be specified as mandatory. This means that a null value is not a valid value for this attribute.

Not changeable

Description

Specifying that the attribute value is not changeable implies that it cannot be updated. It may however be changed to correct erroneous information.

Exhausts value class

Description

This feature allows the database designer to specify that every member of the value set must be the value of some attribute. The attribute Clients_of class AGENT_INFO "exhausts value class" because every client must be the client of some agent.
MEMBER_ATTRIBUTES

Example

AGENT_INFO

description: Attributes that the agent and subagents have in common. Used as a value class for Agent_particulars of AGENT and Subagent_particulars of SUBAGENT.

member attributes:

Clients_of

  value class: CLIENT
  inverse: Agent_of_record
  multivalued
  exhausts value class
  no overlap in values

Licensed_with

  value class: COMPANIES
  multivalued

Number_of_clients

  value class: INTEGERS
  derivation: number of unique members in Client_of

No overlap in values

Description

This feature can be used to specify exclusive ownership of a value by an attribute. The attribute Clients_of, in the above example, is also declared as having "no overlap in value, which means that no two Agents can have the same Client."
INVERSE
======

Syntax

inverse: ATTRIBUTE_NAME

Description

The matching and inverse clauses allow the occurrence of data redundancy within the database. They were created intentionally to support the incorporation of user views into the SDM schema. In the design of a database it sometimes becomes convenient and meaningful to think of an attribute and entities switching places to accommodate different user perspectives. Inverse is used to accommodate such a situation. The attribute Client_of in AGENT_INFO has a value class of CLIENT and its inverse is the attribute Agent_of_record in the class CLIENT which has a value class of AGENT_INFO. Kroenke (6) points out that the inverse clause allows us to describe two entities which are contained within each other. Thus if we know a client, his agent can be determined by the attribute agent_of_record. A different view of the data might present us with an agent whose clients may be determined by the attribute, Client_of.

Examples

AGENT_INFO

description: Attributes that the agent and subagents have in common. Used as a value class for Agent_particulars of AGENT and Subagent_particulars of SUBAGENT.

member attributes:

Clients_of

value class: CLIENT
inverse: Agent_of_record
multivalued
exhausts value class
no overlap in values

Licensed_with

value class: COMPANIES
multivalued

Number_of_clients

value class: INTEGRERS
derivation: number of unique members in Client_of
CLIENT

definition: individuals who purchase products from the enterprise.

member attributes:

  Client_name
    value class: PERSON_NAMES

  Client_personal_info
    value class: PERSONAL_INFO

  Client_spouse_info
    value class: SPOUSE_INFO

  Client_dependent_info
    value class: DEPENDENT_INFO
      multivalued

  Client_business_info
    value class: CLIENT_BUSINESS_INFO

Miscellaneous_information
    value class: MISC_INFO

  Annuities_owned
    value class: ANNUITIES_INFO
      multivalued

  IRA's_owned
    value class: IRA_INFO
      multivalued

  Agent_of_record
    value class: AGENT_INFO
    inverse: Client_of

identifiers: Client_name
MATCH
=====

Syntax

match: ATTRIBUTE_NAME OF CLASS_NAME on ATTRIBUTE_NAME

Description

Where inversion allows the description of binary associations between attributes, matching supports binary and higher degree associations. Matching provides a means of deriving an attribute's value from information within the database by matching common values of attributes from different classes. There is some disagreement as to the interpretation of the match derivation. In this text, match will be used as a method for deriving a single attribute value. The other interpretation is a more powerful join in which a match derives an entire class instance. An example of matching is found in the class PRO_FORMA_CASH_FLOW_STATEMENT. In this example the total insurance premiums paid by a client is derived from the value of the attribute Total_gross_yearly_premiums of the class POLICY_RECORD by a match on the client's name. Matching can be thought of as a high level join. It is used extensively in the insurance agency schema because of the highly redundant nature of the application environment. In this application environment example CLIENT and POLICY_RECORD serve as the primary sources of information. Attribute values of most of the other classes may be derived from these primary classes by matching on the client's name or on a specific policy number. Additional examples may be seen in CLIENT_FILE_CARD, CLIENT_FILE_CARD_POLICY_INFO, and the FOLLOW-UPS (Appendix D).
MATCH
=====

Example

PRO_FORMA_CASH_FLOW_STATEMENT

description: That data stored in the database that would be of use when filling out a client's pro forma cash flow statement.

member attributes:

Client_name
  value class: PERSON_NAMES

Client_address
  value class: ADDRESSES

Client's_total_yearly_insurance_premiums
  value class: DOLLARS
  match: Total_gross_yearly_premiums of POLICY_RECORD on Client_name

identifiers: Client_name
ATTRIBUTE DERIVATIONS
========================

Syntax

MEMBER ATTRIBUTE DERIVATION <-
[INTERATTRIBUTE DERIVATION; MEMBER SPECIFIED DERIVATION]

CLASS ATTRIBUTE DERIVATION <-
[INTERATTRIBUTE DERIVATION; CLASS SPECIFIED DERIVATION]

Description

Attribute values may be derived from other information stored within the database or in fact may take the form of statistics about the data within the database, e.g., maximum, minimum, average and sum. Values for member attributes and class attributes may be derived, and they share common interattribute derivations. They differ in member specified derivations and class-specified derivations as seen in the syntax diagrams.

As a brief review, the diagrams specify that a MEMBER ATTRIBUTE DERIVATION may take the form of an INTERATTRIBUTE DERIVATION or a MEMBER SPECIFIED DERIVATION and a CLASS ATTRIBUTE DERIVATION may take the form of an INTERATTRIBUTE DERIVATION or a CLASS SPECIFIED DERIVATION.
INTERATTRIBUTE_DERIvation
=================================

Syntax

INTERATTRIBUTE_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]

Description

Interattribute derivations allow the database designer extreme
flexibility in specifying the values a derived attribute may have.

Same as MAPPING

Description

This clause allows the database designer to derive the value of an
attribute by stating that it is the same as that for another attribute.

Example

OTHER_APPOINTMENTS
  description: Appointments other then those that are normally
              scheduled.
  member attributes:
    : Date_of_appointment
      value class: DATES
      Next_date_of_action
      description: This attribute must be present if
      OTHER_APPOINTMENTS is to be included in
      DAILY_SCHEDULE. Membership in DAILY_SCHEDULE is
      based on a match with Next_date_of_action. Rather
      than change the name of the attribute
      Date_of_appointment, which is familiar to all
      employees, Next_date_of_action was coined as a
      synonym.
      value class: DATES
      derivation: same as Date_of_appointment
      ;
**INTERATTRIBUTE DERIVATION**

**Subvalue of MAPPING where is in [CLASS NAME: ATTRIBUTE_PREDICATE**

**Description**

We can define an attribute or mapping to be a subset of the values of a multivalued attribute requiring that they be members of some other class or by specifying some predicate which must be satisfied. This derivation could be used to describe a new attribute of AGENT_INFO whose members would include all clients who are members of the class SPECIAL_CLIENT described in the discussion of subclass predicates.

**Example**

Agent'a_special_clients
  description: all of the agent's clients who are members of the class SPECIAL_CLIENT
  value class: CLIENT
  derivation: subvalue of Client_of where is in SPECIAL_CLIENT

**Where is in MAPPING and is in MAPPING: is in MAPPING or is in MAPPING: is in MAPPING and is not in MAPPING**

**Description**

The database designer could use this predicate to define an attribute whose value represents the intersect, union or difference of two other member attributes or mappings. To demonstrate this predicate two additional member attributes of AGENT_INFO must be defined.
INTERATTRIBUTE_DERIVATION
=================================

Examples

Agent's_clients_55_and_over
   description: All of the agent's clients who are members of the class SENIOR_CLIENTS.
   value class: CLIENT
   derivation: subclass of Client_of where is SENIOR_CLIENTS

Agent's_special_clients_55_and_over
   description: An agent's clients who are members of both Agent's_special_clients and Agent's_clients_55_and_over
   value class: CLIENT
   derivation: subvalue of Clients_of where is in Agent's_special_clients and Agent's_clients_55_and_over

= MAPPING_EXPRESSION

Description

An attribute's value can be derived by applying an arithmetic expression to the values of other member attributes or mappings. This type of derivation can be seen in the Date_of_next_action definitions of the "FOLLOW-UP"s (Appendix D). The example demonstrates its use in the description of the class REINSTATEMENT_FOLLOW-UP. Although it is specified that the involved attributes or mappings must have numeric values the SDM syntax has been expanded by the database designer to include the arithmetic manipulation of dates. The various symbols and their meanings are as follows: (+) addition; (-) subtraction; (*) multiplication; (/) division; and (^) exponentiation.
INTERATTRIBUTE_DERIVATION

Example

REINSTATEMENT_FOLLOW-UP
  description: Calls to insurance company to ascertain whether a
  client has paid his premium.
  member attributes:
    Date_of_next_action
    value class: DATES
    derivation: = Date_notified_of_delinquency + 14
      if this is the first attempt
    or Date_of_next_action + 1
      if client was not notified on the initial date,
      denoted by a "no" value of Contact_completed of class
    DAILY_SCHEDULE for this instance
  **** Note: This is not part of the SDM syntax. ****

[M maximum: minimum: average: sum] of MAPPING

Description

"The operators 'maximum', 'minimum', 'average', and 'sum' can be
applied to a member attribute or mapping that is multivalued; the value
class of the attributes involved must be an eventual subclass
NUMBERS. The maximum, minimum, average, or sum is taken over the
collection of entities that comprise the current value of the attribute
or mapping."(1) The SDM syntax was modified by the database designer to
allow these operators to be applied to the name class DOLLARS.
INTERATTRIBUTE_DERIVATION

Examples

Total_gross_yearly_premium
value class: DOLLARS
 derivation: sum of Gross_yearly_premium

DOLLARS
description: The form a reference to money may take.
 interclass connection: subclass of STRINGS where format is
 "\$"
 000 < number <= 999
 "," 
 000 <= number <= 999
 "," 
 000 <= number <= 999
 "." 
 00 <= number <=99
 where leading zeros are omitted

Number of {unique} members in MAPPING

Description

The value of an attribute may be defined as being equal to the
number of members in a multivalued attribute or mapping. This
derivation may count all members or only unique members of a
multivalued attribute. The member attribute Number_of_clients of
AGENT_INFO is used to represent the number of unique clients an agent
has.
INTERATTRIBUTE_DERIVATION
=================================

Example

AGENT_INFO

description: Attributes that the agent and subagents have in common. Used as a value class for Agent particulars of AGENT and Subagent particulars of SUBAGENT.

member attributes:

  Clients_of
    value class: CLIENT
    inverse: Agent_of_record
    multivalued
    exhausts value class
    no overlap in values

  Licensed_with
    value class: COMPANIES
    multivalued

  Number_of_clients
    value class: INTEGERS
    derivation: number of unique members in Client_of
MEMBER-SPECIFIED-DERIVATIONS
================================

Syntax

MEMBER-SPECIFIED_DERIVATIONS <-
  [order by [increasing; decreasing] <MAPPING> {within <MAPPING>};
   if in CLASS_NAME;
   [up to CONSTANT; all] levels of values of ATTRIBUTE_NAME;
   contents]}

Description

Member attributes and class attributes can be defined using interattribute derivations or derivations unique to each attribute type (member attributes or class attributes).

Order by [increasing; decreasing] <MAPPING> {within <MAPPING>}

Description

The ordering derivation allows the definition of an attribute which denotes the sequential position of each specific member of the class with respect to some other attribute or mapping within the class. The member attribute Seniority of the class EMPLOYEE is described as being the sequential position of each employee with respect to the date he began working for the agency. Ordering can be either increasing or decreasing with respect to the attribute value; the default is increasing. Ordering can also take place within a subset of the members of a class by specifying "within MAPPING". This causes the ordering of a subset of the members of the class whose values are the same for the mapping referenced to in the above mentioned predicate. For example, the database designers could have described a seniority attribute that would specify each employee's seniority within his employee type. This would have the effect of assigning a sequential ordering value to each office worker based on his seniority among the office staff, assign a
MEMBER-SPECIFIED-DERIVATIONS
================================

subagents seniority with respect to other subagents, etc.

Examples

Seniority
  value class: INTEGERS
  derivation: order by decreasing Date_joined_company

Seniority_within_type
  value class: INTEGERS
  derivation: order by decreasing Date_joined_company within Status_of.

If in CLASS_NAME

Description

An existence attribute is provided that allows the assignment of a boolean value to a member attribute based on the presence of the member in some other specified class. If the database designer wished to signify for each client whether he was considered a special client (described earlier in the discussion of subclasses) a new member attribute of class CLIENT could be described that would assign the boolean value "yes" if the member in question was also a member of the class SPECIAL_CLIENTS or "no" otherwise.

Example

Is_special
  value class: YES/NO
  derivation: if in SPECIAL_CLIENTS

[up to CONSTANT; all] levels of values of ATTRIBUTE_NAME

Description

To describe the next clause "up to [CONSTANT;ALL] levels of value of ATTRIBUTE_NAME" a new attribute of CLIENT must be defined.
Relatives_who_are_clients will allow us to combine all the entities obtained by recursively tracing the values of Client_dependent_info\(\cdot\)Name. This will provide a list of a client's children, the children's children, etc. The recursive process can be repeated a specified number of times (up to CONSTANT levels) or for all occurrences (all levels). Although the syntax does not specify that a mapping can be used, a mapping is necessary to impart the proper meaning, thus the "\(\cdot\)Name" enclosed in parentheses in the description of the attribute Relatives_who_are_clients. This then tells us that we must trace relatives by identifying those dependents that are also listed as clients. We must use the dependent's name as an identifier for CLIENT.

Example

Relatives_who_are_clients
description: a recursive descent through Client_dependent_info that will produce a list of relatives who are also clients.
value class: CLIENT
derivation: all levels of values of Client_dependent_info \(\cdot\)Name

Contents

Description

The last member-specified_derivation clause, "contents", is stated by Hammer and McLeod (4) to automatically exist when a grouping class is defined. "The value of this attribute is the collection of members (of the class underlying the grouping) that form the contents of that member\(\cdot\)"(4). To demonstrate the use and significance of "contents" we will have to add a member attribute to our definition of the class TYPE_OF_COMPANIES. The attribute Instances has as its members all
MEMBER-SPECIFIED-DERIVATIONS

instances of companies which have the same value for Company_type.

Example

TYPE_OF_COMPANIES
   description: Type of companies represented by the agency.
   interclass connection: grouping of COMPANIES on common
                             value of Company_type groups defined as classes are . .
   member attributes:
     Instances
       description: the instances of the type of companies.
       value_class: COMPANIES
       derivation: same as contents
       multivalued
CLASS_ATTRIBUTE

Syntax

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}

Description

SDM recognizes two attribute types, member attributes (discussed
previously) and class attributes. Member attributes describe properties
belonging to each member of the class and class attributes describe
properties of the class as a whole. Class attributes and member attributes
share most of their formal description in common. The major difference is
in the description of CLASS_SPECIFIED_DERIVATIONS and the lack of the
follow two clauses in the class attribute description, "exhausts value
class" and "no overlap in value".

Descriptions of those clauses shared in common can be found in the
section discussing member attributes. The class-specified derivations will
appear as a major topic and follows next.
CLASS-SPECIFIED DERIVATION
==================================

Syntax

CLASS-SPECIFIED_DERIVATIONS
[ number of {unique} members in this class;
[ maximum; minimum; average; sum] of ATTRIBUTE_NAME over members of this class]

Description

The class-specified derivations allow the database designer to maintain statistical information about the members of the class and attributes of the class. An example of the derivation "number of {unique} members in this class" appears in the description of the attribute Number_of_employees. The second derivation, "[maximum; minimum; average; sum] of ATTRIBUTE_NAME over members of this class" would be applied in the same fashion as its member-specified derivation counterpart. The statistical derivations maximum, minimum, average and sum in the original SDM syntax applied only to attributes with the value class NUMBERS. This restriction has been relaxed in this paper to allow for the summation of attributes with the value class of DOLLARS.

Examples

Number_of_employees
value class: INTEGERS
derivation: number of members in this class
Now that the reader has been introduced to the SDM DdDL, it is time to consider how to apply the SDM to an application domain. The first part of this chapter will provide a list of several steps that in the schema design process a database designer may follow. The designer follows these steps with repeated iteration of some steps to complete the schema.

The second part of this chapter will trace the use of these steps to develop the insurance agency schema. An exhaustive description of these steps would be cumbersome and inappropriate for this report. The steps will be presented in sufficient detail as to provide the reader with an understanding of what they involve.

**Design steps**

No definitive algorithm exists for the design of an SDM schema. McLeod (10) summarizes the design process by describing 3 principle tasks which must be accomplished. They are:

"1. A core of classes containing basic things must be defined.

2. Families of related classes must be formed, using subclasses and grouping primitives . .

3. Associations among things must be expressed, using attribute inversion, attribute matching and classes."

Iteration of these tasks is necessary. These tasks have been further refined by McLeod (11). A summarized list of the steps he suggests is given here:
1. Prepare an initial list of the classes, i.e., entities, events, that are to be present in the SDM database. The list should contain the objects and events (things of interest) that occur in the application environment. A brief description of each class should also be included with the list.

2. Examine the class list to determine subclasses and grouping-classes that exist or are implied by the list. This is accomplished by considering each class in the list and determining if there are any entities that are subclasses of the class in question or if the class in question suggests any abstract or aggregate classes whether as a class or as a member of such a class.

While perusing the class list, the database designer should look for similarities in classes which would suggest that they be collected into a common class. McLeod and King (11) refer to such a collection as a family and define a family in the following manner:

"A family is a set of classes that all have, at some level of abstraction, a common type of member; that is, the members of each component of a family of classes belong to some common class."

The classes agent, subagent, office manager and office personnel, by virtue of common attributes would seem to form a meaningful family called Employee. Employee would thus become the parent class and agent, subagent, office manager and office personnel would be defined as subclasses of Employee. The primary consideration when looking for families of classes is to determine at what level of abstraction parent classes should be defined. Classes which describe fundamentally different things should not be defined as subclasses of the same class. Those classes which share common attributes become likely candidates as subclasses of a common parent class.
The degree of subclass differentiation must also be considered when defining classes and subclasses. For example, defining subagent as a subclass of agent which in turn is a subclass of Employee was considered inappropriate for our model. The database designer found it more appropriate and meaningful to define both as subclasses of the parent Employee. Such decisions are arbitrary and at some point can only be based on the designer's intuition. However, McLeod and King (11) have provided the following suggestion:

"A useful heuristic for determining the appropriate degree of specificity (i.e., differentiation) is that a subclass be defined if and only if one or both of the following are true:

--- the members of the subclass have attributes that are not possessed by all of the members of its parent class.

--- the subclass will often be manipulated as a meaningful unit."

New classes recognized at this point should be described as in step 1.

3. Produce a list of the aspects (attributes) of interest for each class in the class list. Each aspect must be identified as being an aspect of each member (member attributes) or as an aspect of the class as a whole (class attributes). The designer should provide as a minimum, a name for the attribute, a description of its meaning and should specify if the attribute is multi-valued (default is single-valued). Additional descriptors which may be specified at this time include: whether the attribute may have a null value; whether the value of the attribute is changeable; and in the case of a member attribute, whether every member of the underlying value class must be the value of some attribute (exhausts value class); and/or whether a specific value may be shared by other instances of the attribute (no overlap in values). These optional descriptors may be added later as additional insight into the nature of
the application environment is gained.

4. Assign a value class to each attribute identified in step 3. The designer should examine the class list established in step 1 to select the appropriate value class. If no such value class exists, a new class is added to the list and steps 1 through 3 are repeated for it.

5. Determine the interclass connections for the subclasses and groups defined in step 2. Subclass connections may be defined by applying a predicate on the other member attributes of the parent class (attribute-defined subclass), by the predicate "where specified" (user-controllable subclass), by a predicate on members of two other specified database classes (set-operator-defined subclasses) or by a predicate on current values of some attribute of another database class (existence subclass). The database designer must also guard against circular definitions when defining grouping classes. McLeod (11) provides the following recommendations when defining subclasses: 1) guard against circular definitions, e.g., defining a subclass, C1, of class C2 then later defining C2 as a subclass of C1; 2) it is desirable to define a subclass via a multiset operator using other classes that are of interest, this allows the subclass to inherit attributes from all of the classes involved; 3) multiple levels of subclasses should only be defined when the class at each level is of interest.

Grouping connections may be defined as all those classes having a common value for one or more designated member attributes (expression-defined grouping class); by explicitly specifying the classes which are members of the grouping class (enumerated grouping class) or by defining a grouping class that consists of user-controllable subclasses of some underlying class (a user-controllable grouping class). The last two methods differ in that in the enumerated grouping class all members of
those classes which constitute the grouping class are members of the grouping class. Members of a user-controllable grouping are those specific instances of a class which have been specified as being members of the grouping class.

6. Determine whether each attribute of every class is either primitive or derived. The value of a primitive attribute is directly modifiable by users. The value of a derived attribute can not be directly modified by a user. However, the value of only one attribute in an inverse interrelationship is modifiable and no attribute described with a match interrelationship is modifiable.

   Primitive member attributes may be further defined by the optional descriptors discussed in step 3 (e.g., may not be null, not changeable, exhaust value class, no overlap in values). Class attributes may be further defined by the optional descriptors "may not be null" and "not changeable".

   Derived attributes are assigned a derivation specification. Knowledge of the attribute's value class gained in step 4 will assist in selecting the appropriate derivation specification (see INTERATTRIBUTE_DERIVATION, MEMBER-SPECIFIED_DERIVATION AND CLASS-SPECIFIED_DERIVATION of Appendix A and B). If no appropriate derivation type can be found, auxiliary attributes may be defined. The database designer must repeat step 4 for each new attribute defined.

7. Select one or more member attribute to serve as a unique identifier for each base class if appropriate.

8. Identify all name classes, define each as subclass of either STRINGS or NUMBERS and apply derivation specifications if desired to further describe the class.
In brief, the steps described above can be summarized as follows:

1. Prepare an initial list of the entities (class list).
2. From the class list determine the subclasses and grouping classes of interest.
3. Identify member attributes and class attributes.
4. Assign a value class to each attribute.
5. Determine the interclass connection for subclasses and grouping classes.
6. Add descriptive options to primitive attributes, assign attribute interrelationship specifications if appropriate or and assign derivation specifications to derived attributes.
7. Select member attributes to serve as unique identifiers of appropriate base class.
8. Identify name classes, define them as subclasses of either STRING, INTEGER or NUMBERS, and apply derivation specifications as desired.

As new classes and attributes emerge, repeat the appropriate steps for each.

Application of the design steps

The remainder of this chapter will follow the evolution of the SDM schema for the insurance agency described in Chapter 3. The resulting schema appears in Appendix D. An exhaustive review of the design phase is impossible thus the following discussion will focus on the problems encountered. The reader is encouraged to examine Appendix D, refer to Appendix A and B, and study the discussion of the SDM syntax in Chapter 3 for a more thorough understanding of SDM.
Step 1

1. Prepare an initial list of classes of interest.

On-site interviews and examination of the forms used by the enterprise revealed an initial set of entities and events which appear in figure 5-1.

Entities

Agent  Client
Subagents  Payment
Office manager  Client asset sheet
Office staff  Client file card
Insurance  Estate settlement worksheet
Annuities  Client courtesy cards
IRA's  Quarterly state of business report
Companies  Annual state of business report
Product license  Monthly cash-flow
Commission, subagent
Commission, agent

Events

Appointments  Client follow-up
Reinstatement follow-up  Replacement follow-up
Prospect letter follow-up  Annual reviews
Daily contacts

Figure 5-1. A list of initial entities and events collected from on-site interviews and examination of forms used by the insurance agency.

Step 2

2. From the class list determine the subclasses, grouping classes and class families of interest.

An interesting phenomenon which is obvious but often overlooked is the fact that as we discuss objects and events, we naturally tend to group like "things" together. We can use this phenomenon to our advantage during the on-site visit and while perusing the forms the enterprise uses. While compiling
the entity-event list (figure 5-1) the designer found that related "things" seemed to be grouped and such groups can be used when proceeding with step 2.

The first four entities (agent, subagent, office manager and office staff) suggested a family of classes which represent the Employees of the enterprise. This appears to be a convenient, natural grouping both intuitively and because they are likely to share common attributes. The designer must now decide the most appropriate level of abstraction in which to model this relationship.

One method of representing this relationship would be to define a new class that we shall call EMPLOYEE and define agent, subagent, office manager and office staff as subclasses of EMPLOYEE. We could then differentiate the various members of each subclass by specifying a unique value of some classification attribute (figure 5-2).

A second method of representing this relationship would be to treat each type of employee as members of a common class and assert their relationship, one to another, by describing them as members of an overlaying grouping class. Such an approach would provide the alternative represented in figure 5-3.
EMPLOYEE
description all people who work for the agency.
member attributes:
:
  Status_of
  value class EMPLOYEE_TYPES
  :

AGENT
description: The owner and primary agent of the enterprise.
interclass connection: subset of EMPLOYEE where
  Status_of = "Agent"
  :

SUBAGENT
description: people licensed to sell products and work for the primary agent.
interclass connection: subset of EMPLOYEE where Status_of = "subagent"
member attributes:
:
:

Figure 5-2. Agent, subagents, office manager and office_staff described as subclasses of EMPLOYEE (attribute-defined subclasses).

EMPLOYEE
description: all people who work for the agency.
member attributes:
:
  Status_of
  value class EMPLOYEE_TYPES
  :

EMPLOYEE_TYPES
description: groups of employees by type.
interclass connection: grouping of EMPLOYEE on common value of Status_of.
member attributes:
:
:

Figure 5-3. Agent, subagents, office manager and office_staff implicitly described as groups of employees whose instances are the members of the groups that make up EMPLOYEE_TYPES (expression-defined grouping).
EMPLOYEE_TYPES could be defined as an expression-defined grouping class which would group the various employee types of the base class EMPLOYEE with respect to common values of a specific attribute of EMPLOYEE, e.g. Status_of. In this case the addition of a new employee type to the schema would require no further definition. All employees would then belong to the base class EMPLOYEE and nothing would distinguish one type from another except the value of the Status_of attitude.

A second example involving levels of abstraction is seen in the consideration of the entity Subagent. Should subagent be included within the base class EMPLOYEE as in figure 5-3? Should it be defined as a subclass of EMPLOYEE as in figure 5-2, or would it be more appropriate to define Subagent as a subclass of the subclass Agent? Is office manager a subclass of office staff? Because of the number of attributes each employee type have in common, it seems more natural for at least some of them to be modeled as subclasses of the parent class EMPLOYEE (figure 5-2). If we apply the heuristics mentioned in step 2 we might conclude that based on the difference in the way commissions are calculated Agent and Subagent should be treated as separate subclasses of the parent class EMPLOYEE. Because the attributes which describe office staff and office manager are the same, these two subclasses will be consolidated under office staff and the office manager will be distinguished in some other manner.

The next group of classes on the list --- insurance, annuities, IRA's --- are sufficiently different that they should be treated as separate base classes. However, they suggest a higher level grouping class that would be convenient if one wishes to treat all products offered by the enterprise as a single unit, e.g., total value of products sold. Collecting these classes into a grouping class would seem appropriate, however, none of the clauses available to use for defining a grouping class seem to fit. All grouping
predicates require that the classes constituting its description be defined as subclasses of some parent class. So if we wish to treat these three classes as a single unit, we cannot define them as separate base classes. It was decided that the agency desired information about these products only with respect to what individual clients owned and general information about products as a whole was not needed. Thus they were treated as value classes for attributes describing CLIENT.

A first assessment of the remaining classes on our list appears in figure 5-4. The results of applying step 2 to the initial list of classes appears in figure 5-5.

*Step 3*

1. Identify member attributes and class attributes.

The database designer must now produce a list of the attributes for the classes listed above. Attributes of interest for the forms and reports can be determined directly from these articles. Most of the clients attributes can be ascertained from the client file card. Additional attributes of interest will have been obtained from those interviewed during the on-site visit. From these resources an initial list of attributes for each class can be produced. After this step a second visit to the enterprise should be planned to determine the validity of assumptions made up to this point and to discuss other issues which may have arisen.

Appendix C contains the results of step 3. Class descriptions have been included but attribute descriptions were intentionally omitted to elucidate each class and its attributes. It is important to include class and attribute descriptions as you proceed with the SDM schema. The designers soon realized that the descriptions offered in this Appendix C were inadequate when attempting to continue with the next steps in the design process. As we
proceed with the construction of the SDM schema the absolute necessity of
comprehensive class descriptions (and attribute descriptions when appropriate)
was realized. The grand design that the designers had in mind when the classes
were identified was no longer self evident when it became time to piece the
puzzle together. It was only after time was taken to regroup and, for all
practical purposes, start again beginning with the list prepared in step 1
(figure 5-1) were the designers able to progress toward the completion of the
schema.

Company -- base class

Product license -- value class of an attribute describing the
products an agent is licensed to sell

Commission, agent and Commission subagent -- probably derived
attributes of agent and subagent respectively.

Client -- base class

Payment -- subclass of events

Client asset sheet, client file card, cash flow statement, pro forma
cash flow statement, policy record, statement of financial
position, life insurance form and estate settlement worksheet
-- forms used by the enterprise. Much of the data gathered on
these forms are redundant and much is not needed within the
database. The decision as to how much should be stored is a
function of memory storage available and the frequency of use
of each data item (data field). It was decided that the Policy
record form and the Client file card are the only forms to be
incorporated into the database. Data stored within the database
which could be used by individuals when completing the other
forms will be modeled within the database as summaries of each
form.

Daily contact report -- this report is the event scheduler of the
enterprise. All events listed have a close relationship to this
report. (The name was later changed to DAILY_SCHEDULE.)

Appointments, Client follow-up, reinstatement follow-up, replacement
follow-up, Prospect letter, Prospect letter follow-up, Annual
review, Daily contacts -- subclasses of the base class event.

Quarterly state of business, Annual state of business, monthly cash
flow -- subclasses of the base class event.

Figure 5-4: An initial assessment of the entities and events appearing in
figure 5-1.
<table>
<thead>
<tr>
<th>Base classes</th>
<th>Subclasses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>Appointment</td>
</tr>
<tr>
<td>Products</td>
<td>Client follow-up</td>
</tr>
<tr>
<td>Companies</td>
<td>Reinstatement follow-up</td>
</tr>
<tr>
<td>Product license</td>
<td>Prospect letter</td>
</tr>
<tr>
<td>Client</td>
<td>Annual review</td>
</tr>
<tr>
<td>Policy record</td>
<td>Daily contacts</td>
</tr>
<tr>
<td>Client file card</td>
<td>Agent</td>
</tr>
<tr>
<td>Client asset sheet</td>
<td>Subagent</td>
</tr>
<tr>
<td>Client file card</td>
<td>Office staff</td>
</tr>
<tr>
<td>Cash flow statement</td>
<td>Insurance</td>
</tr>
<tr>
<td>Pro forma cash flow statement</td>
<td>Annuities</td>
</tr>
<tr>
<td>Policy record</td>
<td>IRA's</td>
</tr>
<tr>
<td>Statement of financial position</td>
<td>Payment</td>
</tr>
<tr>
<td>Life insurance form</td>
<td>Client asset sheet summary</td>
</tr>
<tr>
<td>Estate settlement worksheet</td>
<td>Cash flow statement summary</td>
</tr>
<tr>
<td></td>
<td>Pro forma cash flow statement</td>
</tr>
<tr>
<td></td>
<td>statement</td>
</tr>
<tr>
<td></td>
<td>Statement of financial position</td>
</tr>
<tr>
<td></td>
<td>summary</td>
</tr>
<tr>
<td></td>
<td>Life insurance summary</td>
</tr>
<tr>
<td></td>
<td>Estate settlement worksheet</td>
</tr>
<tr>
<td></td>
<td>summary</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Grouping Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commission, agent</td>
<td>none identified</td>
</tr>
<tr>
<td>Commission, subagent</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5-5. A list of entities and events described in terms of SDM categories.
Steps 4-6

4. Assign a value class to each attribute.
5. Determine the interclass connection for subclasses and grouping classes.
6. Add descriptive options to primitive attributes, assign attribute interrelationship specifications if appropriate or and assign derivation specifications to derived attributes.

The assignment of value classes, determining interclass connections and assigning derivation specifications claimed the bulk of the time required to design the SDM schema. All three steps resulted in the discovery of additional classes, the realization that some were not needed and the need to restructure others. Steps 2 through 6 were repeated numerous times as new avenues of expression were explored. In addition to the insight gained by creating the enhanced class and attribute descriptions, "clutter" was removed from the model by the decision to place commonly referenced groups of attributes into their own classes.

An examination of Appendix C will reveal a number of classes that share common groups of attributes (at least they have the same attribute names). The SDM model allows the use of non-unique attribute names as long as they occur in different class hierarchy. In other words, attributes with the same names cannot be related by grouping classes or subclasses. We have not determined the interclass connections for our model as yet but having just compiled a list of classes and their associated attributes we have an idea of the attributes that are identical within different class definitions. It is both cumbersome and somewhat ambiguous to repeatedly reference each individual attribute within these groups when we describe a class. The group of attributes consisting of Name_of_agent, Client_name, Policy_name and Policy_type is used in the description of five different classes (APS_FOLLOW-UP, CLIENT_FOLLOW-UP, OUT_OF_TOWN_FOLLOW-UP, REINSTATEMENT_FOLLOW-UP, and REPLACEMENT_FOLLOW-Up). It is obvious to those currently involved with the construction of this conceptual model that the attributes are the same in all
five descriptions, this is not to say that they represent the same instances in all five classes, but rather, they share the same value class. However, a new applications programmer or a new employee wishing to understand the function and relationships of "things" within the enterprise will not share such insight. Recognizing that these groups of attributes appear frequently and wishing to exploit SDM's capabilities to impart meaning into a conceptual schema the database designers decided to place commonly referenced groups of attributes into their own classes. Now when a reference is made to the four attributes, Name_of_agent, Clients_name, Policy_type, and Company, a new attribute description is used with a value class of POLICY_INFO_FOR_FOLLOW-UP which will contain the above mentioned attributes. As a result of collecting commonly used attributes into their own class six "attribute" classes were added to our schema. The new classes are described at the bottom of Appendix C and are proceeded by a number. The number corresponds to the number enclosed in parentheses behind attributes which were initially defined in other classes and are now part of the new class definitions.

It is important to remember that the "attribute" class was introduced to conceptually "save space" in the schema and to enhance the understanding of these attributes and the role they play in the schema. We save space by placing attributes which are commonly referenced together into a single class. We enhance understanding by emphasizing the fact that they are indeed the same attributes that are being referenced in several class definitions. The designers and users of the model must be comfortable with the fact that attributes which were partitioned into "attribute" classes are still part of the original class description in which they now appear as a value class. If this detracts from the understanding of the model then the members of the "attribute" class should be moved back into the original class descriptions.

Although the addition of attribute classes clarified some aspects of the
schema it complicated the match derivation. The syntax for the match derivation is as follows:

match: ATTRIBUTE_NAME of CLASS_NAME on ATTRIBUTE_NAME

Mapping and a type of reverse mapping were needed to accommodate matching on attributes that are members of an attribute class. A portion of the description of the class INSURANCE_APPLICATION_INFO_SUMMARY demonstrates the extended matching derivation (figure 5-6).

The client's home address (Client_address) is contained within the database and is derivable by assuming the value of the attribute Home_address which is a member of the attribute class PERSONAL_INFO (figure 5-7). PERSONAL_INFO is the value class of Client_personal_info which is an attribute of CLIENT (figure 5-8). The description of CLIENT also contains Client_name as an attribute. Thus by matching Client_name of INSURANCE_APPLICATION_INFO_SUMMARY with Client_name of CLIENT the attribute Client_address may be obtained. This match description is also represented in a diagram in figure 5-9. In this match example the class CLIENT is specified as the source of the information and a mapping is used to specify the attribute containing the information ("match: Client_personal_info.Home_address of CLIENT on Client_name").

The description of Spouse_name (figure 5-7) demonstrates another match approach. This value also resides within the database and may be obtained by knowing the name of the client whose spouse's name we seek. The same attribute class as above is being queried, however, it is named explicitly as the source of the information in a reverse manner. The phrase "Client_name (of CLIENT)" is a form of reverse mapping that tells us where to find Client_name
INSURANCE_APPLICATION_INFO_SUMMARY

description: That data stored in the database that would be of use when filling out an insurance application form for a current client.

member attributes:

Client_name
  value class: PERSON_NAMES

Client_address
  value class: ADDRESSES
  match: Client_personal_info.Home_address of CLIENT on Client_name

Spouse_name
  value class: PERSON_NAMES
  match: Name_of_spouse of SPOUSE_INFO on Client_name (of CLIENT)

Spouse_date_of_birth
  value class: DATES
  match: Client_spouse_info.Date_of_birth of CLIENT on Client_name

Dependents_name
  value class: PERSON_NAMES
  match: Name_of_dependent of DEPENDENT_INFO on Client_name (of CLIENT)

Dependents_date_of_birth
  value class: DATES
  match: Client_dependent_info.Date_of_birth of CLIENT on Client_name

Spouse_amount_insurance

description: If it is assumed that if the spouse is also a client then a record of total insurance owned should also be present for the spouse. This value is derivable by matching Spouse_name to Client_name in CLIENT_FILE_CARD.

  value class: DOLLARS
  match: Total_insurance_owned of POLICY_RECORD on Spouse_name

Figure 5-6. Partial description of the class INSURANCE_APPLICATION_INFO_SUMMARY excerpt from Appendix D.
PERSONAL_INFO

description: The personal information relevant to each person, both client and employee.

member attributes:
   Home_address
     value class: ADDRESSES
SSN
     value class: SOCIAL_SECURITY_NUMBER
Phone_number
     value class: PHONE_NUMBERS
Date_of_birth
     value class: DATES

Figure 5-7. Description of the class PERSONAL_INFO excerpt from Appendix D.

CLIENT

definition: individuals who purchase products from the enterprise.

member attributes:
   Client_name
     value class: PERSON_NAMES
Client_personal_info
     value class: PERSONAL_INFO
Client_spouse_info
     value class: SPOUSE_INFO
Client_dependent_info
     value class: DEPENDENT_INFO

Figure 5-8. Partial description of the class CLIENT excerpt form Appendix D.

SPOUSE_INFO

description: That information relevant to a client's or employee's spouse.

member attributes:
   Name
     value class: PERSON_NAMES
Home_address
     value class: ADDRESSES
Phone_number
     value class: PHONE_NUMBERS
Date_of_birth
     value class: DATES

Figure 5-10. Description of the class SPOUSE_INFO excerpt from Appendix D.
because it is not an attribute of the class SPOUSE_INFO (figure 5-10). This match description is also represented in the diagram in figure 5-9. The above examples of the match derivation were adapted for use with attribute classes and were thus extensions to the SDM syntax offered by Hammer and McLeod (4).

---

Figure 5-9: Illustration of a match attribute interrelationship, attributes are enclosed in "< >" and classes are enclosed in boxes. The values of the attributes Home_address and Spouse_name in the class INSURANCE_APPLICATION_INFO_SUMMARY are derived by matching the values of Client_name. The derivation path leads from an attribute through its value class to a specific attribute of the value class.

Another peculiarity was encountered in the description of Spouse_amount_insurance (figure 5-6), however, this was not due to the use of attribute classes. If the spouse is also a client then the amount of insurance owned by the spouse which was sold by this agency should also be derivable. A match is specified which would obtain the value of the attribute
Total_insurance_owned of the class POLICY_RECORD (figure 5-12) by matching the value of Spouse_name with its representation (the attribute Client_name) within POLICY_RECORD. This match description is represented in the diagram in figure 5-11. One must be well versed in the syntax of the match derivation to fully understand the intended match.

```
+---------------------+            +---------------------+
 | INSURANCE_APPLICATION_INFO_SUMMARY |            | POLICY_RECORD |
 +---------------------+            +---------------------+
          /            /                        /
 < Spouse_name >     < Spouse_amount_insurance >           /
      |                     |                          |
   match       derive      </ Client_name >     < Total_insurance Owned >
      |                     |                          |
          /            /
 +---------------------+            +---------------------+
```

Figure 5-11: Illustration of a match attribute interrelationship, attributes are enclosed in "< >" and classes are enclosed in boxes. The value of the attribute Spouse_amount_insurance in the class INSURANCE_APPLICATION_INFO_SUMMARY is derived by matching the value of Spouse_name to a value of Client_name of the class POLICY_RECORD. This particular approach to deriving an attribute's value is unusual because of the match of dissimilar attribute names.

The database designers were also unable to develop a meaningful description of DAILY_SCHEDULE (figure 5-13) and Contact_record of CLIENT_FILE_CARD(figure 5-14) using the SDM syntax provided by Hammer and McLeod [4]. The daily schedule was to contain a reference to all events, e.g., appointments, follow-ups, etc., for each day. To do this, values from the above mentioned classes had to be collected based on the current day's
POLICY_RECORD

description: a collection of specifications about insurance policies 
or owned by each client. The POLICY_RECORD class will act as the master 
record of each policy sold.

member attributes:
- Client_name
  - value class: PERSON_NAMES
    :
    :

- Total_insurance_owned
  - value class: DOLLARS
  - derivation: sum of Face_amount
    :
    :

Figure 5-12. Partial description of the class POLICY_RECORD
excerpt from Appendix D.

DAILY_SCHEDULE

description: A schedule of events (follow-ups, appointments, etc.) that 
are to occur on the current day.

member attributes:
- Event
  - value class: APS_FOLLOW-UP, REINSTATEMENT_FOLLOW-UP, 
    PROSPECT_FOLLOW-UP, REPLACEMENT_FOLLOW-UP, OUT_OF_TOWN_FOLLOW-UP, 
    ANNUAL_REVIEW, CLIENT_COURTESY_CARD, OTHER_APPOINTMENTS
  - derivation: where Date_of_next_action = Today's_date
    ### Note: This is not part of the SDM syntax. ###

- Today's_date
  - value class: SYSTEM_DATE
    :
    :

Figure 5-13. Partial description of the class
DAILY_SCHEDULE, excerpt from Appendix D.
date. Originally these classes were to be described as subclasses of some
overlapping parent class. However, this approach became extremely complicated
when it came to describing various scheduling criteria and dealing with
different attribute names which essentially described the same attributes,
e.g., Date_of_next_action. It was more meaningful to describe each event as a
separate base class thus allowing the use of similar attribute names.
Subsequently, one can specify multiple value classes for the attributes Event
of DAILY_SCHEDULE and finally gather all events scheduled for the current day
by the derivation "where Date_of_next_action = Today's_date". A similar
procedure is used in the description of Client_record of CLIENT_FILE_CARD
where all events involving a single client are collected by the derivation
"where Client_name = Client_name (of the appropriate class)".

CLIENT_FILE_CARD

description: a summary of the client attributes, history of
client contacts and a summary of POLICY_RECORD. Most
attributes pertaining to insurance will be derived from
attributes of POLICY_RECORD based on a match with
Policy_number of CLIENT_FILE_CARD POLICY_INFO. Those
attributes pertaining to the client will be derived from
attributes of CLIENT based on a match with Client_name.
member attributes:
Client_name
  value class: PERSON NAMES
  :
Contact_record
  value class: APS_FOLLOW-UP, REINSTATEMENT_FOLLOW-UP,
    PROSPECT_FOLLOW-UP, REPLACEMENT_FOLLOW-UP,
    OUT_OF_TOWN_FOLLOW-UP, ANNUAL_REVIEW,
    CLIENT_COURTESY_CARD, OTHER_APPOINTMENTS
match: Event of DAILY_SCHEDULE on Client_name (of
appropriate class)
**** Note: This is not part of the SDM syntax. ****

Figure 5-14. Partial description of the classes
DAILY_SCHEDULE and CLIENT_FILE_CARD excerpt from
Appendix D.
Steps 7-8

7. Select member attributes to serve as unique identifiers of appropriate base class.
8. Identify name classes, define them as subclasses of either STRING, INTEGER or NUMBERS, and apply derivation specifications as desired.

The selection of identifiers for base Classes is mostly straightforward, however the use of attribute classes did cause a problem. The use of an attribute class (class 6 of Appendix C) to gather common attributes of events resulted in the identifier, Client_name, for the events being placed in the attribute class POLICY_INFO_FOR_FOLLOW-UP. At some later date, it may be desirable to collect all events scheduled for a specific agent on a given day, thus Name_of_agent is another potentially useful attribute which was also placed into POLICY_INFO_FOR_FOLLOW-UP. Rather than introduce a mapping as an identifier, it was decided that the class POLICY_INFO_FOR_FOLLOW-UP be deleted and its 5 attributes would be returned to the event descriptions from which they came. These classes appear in Appendix C and have certain attribute names which are followed by the number 6.

Identifiers are optional and it was thus decided that identifiers for the "attributes" classes discussed previously were not appropriate. The description of name classes is also straightforward and will not be elaborated upon. However, the description of name classes and the selection of identifiers should not be necessarily postponed until last.

Summary

The design steps provide some guidelines for applying the DbDL to the application domain. It was immediately recognized that exhaustive descriptions were essential to the schema's successful completion. Attribute classes provided a convenient and conceptually sound means of representing commonly referenced groups of attributes. Discovered shortcomings include:
1) The SDM's lack of facilities for manipulating times and dates; 2) The lack of a convenient means for grouping dissimilar entities; and 3) A need for some type of automated tool to assist the designer in the application process.
Chapter 6

Converting an SDM model into a relational model

This chapter will briefly outline the steps that were taken to convert the insurance agency schema into a relational model. Because the focus of the report is on the development of a conceptual model, only a few paragraphs will be spent discussing this conversion.

We assume that the system will be implemented on a microcomputer. With this as a guide our ultimate objective will be to minimize data storage requirements. Our initial implementation will minimize the amount of data redundancy within the database. At a later date if it is determined that certain redundancy is necessary to improve efficiency, the implementation will be modified. With this criteria in mind we will proceed with the conversion.

In order to reduce the bewilderment fostered by 26 pages of SDM schema (Appendix D) we will eliminate consideration of all derived attributes. They will be calculated or otherwise retrieved from the database by the applications programs which will access the database. If you are interested in achieving domain/key normal form (8), the name classes may be rewritten as domain definitions. We will not attempt a domain/key representation of this application environment, thus the name classes may be set aside as external documentation. They will not be directly represented within the model.

Having eliminated all derived attributes and name classes, we find that the number of classes and attributes with which we must deal is greatly reduced. A list of the remaining classes appears in figure 6-1.

All base classes which were not attribute classes (those classes containing groups of commonly referenced attributes) become relations. All single valued attributes that are not derived become tuples within the relation (figure 6-2).
Figure 6-1: Classes remaining after eliminating derived attributes.
<table>
<thead>
<tr>
<th>DEPENDENT_INFO</th>
<th>PERSONAL_INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Home_address</td>
</tr>
<tr>
<td>Date_of_birth</td>
<td>SSN</td>
</tr>
<tr>
<td></td>
<td>Phone_number</td>
</tr>
<tr>
<td></td>
<td>Date_of_birth</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>POLICY_RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee_name</td>
<td>Client_name</td>
</tr>
<tr>
<td>Employee_personal_info</td>
<td>Policy_number</td>
</tr>
<tr>
<td>Employee_spouse_info</td>
<td>Type</td>
</tr>
<tr>
<td>Employee_dependent_info</td>
<td>Company</td>
</tr>
<tr>
<td>Date_joined_company</td>
<td>Face_amount</td>
</tr>
<tr>
<td>Status_of</td>
<td>Date_of_issue</td>
</tr>
<tr>
<td>Pay</td>
<td>Age_of_issue</td>
</tr>
<tr>
<td>identifiers: Employee_name</td>
<td>Primary_beneficiary</td>
</tr>
<tr>
<td></td>
<td>WP_rider</td>
</tr>
<tr>
<td></td>
<td>Dividend_option</td>
</tr>
<tr>
<td></td>
<td>Gross_yearly_premium</td>
</tr>
<tr>
<td></td>
<td>Add_rider</td>
</tr>
<tr>
<td></td>
<td>Cpd_rider</td>
</tr>
<tr>
<td></td>
<td>Apl_rider</td>
</tr>
<tr>
<td></td>
<td>Contingent_beneficiary</td>
</tr>
<tr>
<td></td>
<td>identifiers:</td>
</tr>
<tr>
<td></td>
<td>Client_name</td>
</tr>
<tr>
<td></td>
<td>+ Policy_number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRA_INFO</th>
<th>PROSPECT_FOLLOW-UP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Name_of_prospect</td>
</tr>
<tr>
<td>IRA_identification</td>
<td>Prospect_phone</td>
</tr>
<tr>
<td>IRA_value</td>
<td>Contact_made</td>
</tr>
<tr>
<td></td>
<td>Date_of_contact</td>
</tr>
<tr>
<td></td>
<td>identifiers: Name_of_prospect</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOAN_INFO</th>
<th>PROSPECT_LETTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loan_amount</td>
<td>Name_of_agent</td>
</tr>
<tr>
<td>Loan_interest</td>
<td>Name_of_prospect</td>
</tr>
<tr>
<td></td>
<td>Address_of_prospect</td>
</tr>
<tr>
<td></td>
<td>Type_of_product</td>
</tr>
<tr>
<td></td>
<td>Date_mailed</td>
</tr>
<tr>
<td></td>
<td>identifiers: Name_of_prospect</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OTHER APPOINTMENTS</th>
<th>REINSTATEMENT_FOLLOW-UP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name_of_agent</td>
<td>Client_name</td>
</tr>
<tr>
<td>Client_name</td>
<td>Policy_number</td>
</tr>
<tr>
<td>Date_of_appointment</td>
<td>Date_of_next_action</td>
</tr>
<tr>
<td>Time_of_appointment</td>
<td></td>
</tr>
<tr>
<td>Description_of_appointment</td>
<td></td>
</tr>
<tr>
<td>Next_date_of_action</td>
<td></td>
</tr>
<tr>
<td>identifiers:</td>
<td></td>
</tr>
<tr>
<td>Client_name</td>
<td></td>
</tr>
<tr>
<td>+ Date_of_appointment</td>
<td></td>
</tr>
</tbody>
</table>

| PERSON_NAMES | | |
|--------------|---------------|
| Title        | REINSTATEMENT_FOLLOW-UP |
| First        | Client_name    |
| Middle       | Policy_number  |
| Last         | Date_of_next_action |
|              | Date_notified_of_delinquency |
|              | identifiers: Client_name |
Next we will consider the best representation of the attribute classes. We may 1) move all attributes described with an attribute class back into their original parent class; 2) represent the value class of a multivalued attribute as a separate relation; 3) represent multivalued attributes within an attribute class as their own relations and return the remaining attributes to their parent class; represent the attribute class as a separate relation even though it is not the value of a multivalued attribute.

The attributes of attribute classes which are single valued will be placed back into their parent class. These attributes are denoted by having the attribute class name from which they came as part of their name, e.g., PERSONAL_INFO.PERSON_NAMES.First, LOAN_INFO.Loan_amount. The class PERSON_NAME, ADDRESS, LOAN_INFO, CASH_VALUE_AT_AGE, CASH_VALUE_FIRST_DATE and CASH_VALUE_SECOND_DATE exemplify this type of attribute class. Examples of value classes that became separate relations include ANNUITY_INFO, IRA_INFO and DEPENDENT_INFO. Individual multivalued attributes that were placed into their own relations include: Licensed_with (of AGENT_LICENSED_WITH), Client_name (of AGENT'S_CLIENTS) and Products_offered (of COMPANY_PRODUCTS_OFFERED). Other single valued attribute classes which were left as separate relations were SPOUSE_INFO, CLIENT_BUSINESS_INFO, and PERSONAL_INFO. You will notice that in all situations when relations other than the major base classes were constructed, identifiers had to be included within the tuples.

It can be seen from this example that converting the SDM schema to a
relational model was not that complicated. If we had been concerned about normal forms the procedure would have required additional attention. (Second normal form was achieved in this demonstration.)

Figure 6-2: Insurance agency relational model.
OUT_OF_TOWN_FOLLOW-UP(Client_name,Date_policy_sold,Name_of_agent,
    Policy_number,Date_of_next_action,)

POLICY_RECORD(Policy_number,Client_name,Type,Company,Face_amount,
    Date_of_issue,Age_of_issue,Primary_beneficiary,WP_rider,
    Dividend_option,Gross_yearly_premium,Add_rider,Cpd_rider,Apl_rider,
    LOAN_INFO.Loan_amount,LOAN_INFO.Loan_interest,CASH_VALUE_AT_AGE.Age,
    CASH_VALUE_AT_AGE.Guaranteed_value,CASH_VALUE_AT_AGE.Total_value,
    CASH_VALUE_FIRST_DATE.Date,CASH_VALUE_FIRST_DATE.Guaranteed_value,
    CASH_VALUE_FIRST_DATE.Total_value,CASH_VALUE_SECOND_DATE.Date,
    CASH_VALUE_SECOND_DATE.Guaranteed_value,
    CASH_VALUE_SECOND_DATE.Total_value,Contingent_beneficiary,
    CLIENT_FILE_CARD_POLICY_INFO.Mode_of_premium_payment)

PROSPECT_FOLLOW-UP(Name_of_prospect,Date_of_contact,Prospect_phone,
    Type_of_product,Contact_made,)

PROSPECT_LETTER(Name_of_prospect,Date_mailed,Name_of_agent,
    PERSON_NAMES.Title,
    PERSON_NAMES.Name_First,PERSON_NAMES.Name_Middle,
    PERSON_NAMES.Name_Last,ADDRESSES.House#,Apt#,ADDRESSES.Street_Route,
    ADDRESSES.State,ADDRESSES.Zip_code,Type_of_product,)

REINSTATEMENT_FOLLOW-UP(Client_name,Policy_number,
    Date_of_next_action,Date_notified_of_delinquency)

REPLACEMENT_FOLLOW-UP(Client_name,Policy_number,Date_of_next_action)

AGENT'S_CLIENTS(Agent_name,Client_name)

AGENT_LICENSED_WITH(Agent_name,Licensed_with)

ANNUITY_INFO(Name_of_owner,Company,Annuity_identification,
    Annuity_value)

CLIENT_BUSINESS_INFO(Client_name,Occupation,Phone_number,
    ADDRESSES.House#,Apt#,ADDRESSES.Street_Route,ADDRESSES.State,
    ADDRESSES.Zip_code)
DEPENDENT_INFO(\texttt{Parent\ name, \textit{PERSON\ NAMES.title, \textit{PERSON\ NAMES.First, \textit{PERSON\ NAMES.Dependent\ Middle, \textit{PERSON\ NAMES.Dependent\ Last, Date\ of\ birth}}}})

IRA_INFO(\texttt{Owner\ name, Company, IRA\ identification, IRA\ value})

PERSONAL_INFO(\texttt{Name, \textit{PERSON\ NAMES.Title, \textit{PERSON\ NAMES.Name\ First, \textit{PERSON\ NAMES.Name\ Middle, \textit{PERSON\ NAMES.Name\ Last, ADDRESSES.House\#\_Apt\#, \textit{ADDRESSES.Street\ Route, ADDRESSES.State, ADDRESSES.Zip\ code, SSN, Phone\ number, Date\ of\ birth}}}})

SPOUSE_INFO(\texttt{Client\ name, Name, \textit{PERSON\ NAMES.Title, \textit{PERSON\ NAMES.Spouse\ First, \textit{PERSON\ NAMES.Spouse\ Middle, \textit{PERSON\ NAMES.Spouse\ Last, ADDRESSES.House\#\_Apt\#, ADDRESSES.Street\ Route, ADDRESSES.State, ADDRESSES.Zip\ code, Phone\ number, Date\ of\ birth}}}})

\textbf{Figure 6-2 continued}
This final chapter will review several of the ideas discussed in previous chapters as well as present suggestions for an automated SDM design tool. The SDM DbDL proved to be an adequate tool for modeling most of the situations encountered within this application domain. However, several shortcomings were discovered that are worthy of further discussion.

The SDM does not support the manipulation of dates. In order to increment a date by a constant, the date must be stored as a number which implies Julian dates. Julian dates are not natural in a business environment. This then would require the description of a function that would convert Gregorian dates to Julian dates, add the appropriate time increment and convert back to Gregorian dates. The aforementioned function would have to be modified to allow for applications which dealt with update increments of less than 24 hours. Another problem arises when one wishes to model the incremental updates in working days rather than simply n days in the future.

SDM does not provide adequate constructs for the aggregation of dissimilar time and date attributes into a sequence of events. The various follow-ups, the client annual review, and appointments could not be collected together as a simple multivalued attribute. Grouping predicates were inadequate in that they required all members of the aggregation to be defined as subclasses of an underlying class. This stipulation then required that unique names be given to the various follow-up attributes which denoted the date of next action. With different attribute names no derivation could be found within the syntax defined that would allow the integration of the dates into a common list. If all follow-ups were defined in one class and thus being differentiated from one another by the value of a member attribute, the
date of next action for each follow-up could thus be defined as another member attribute. A class attribute could then be defined as a derivation of the date of next action ordered by date or equal to a specified date but there is no way to selectively update the date of next action with the appropriate time increment.

When a situation is encountered that cannot be adequately modeled by strictly following the DbDL syntax, there are two courses of action that can be taken: 1) Construct the model up to the point of conflict, define the attributes and/or classes involved, and describe the circumstances or actions that must be applied to the attributes within the description clause; 2) Define an extension to the DbDL syntax. In most cases the unusual situations which cannot be modeled by SDM will be an application's consideration, not a database consideration.

The present DbDL syntax forces the db designer to use the same name in both classes for the matching attribute. In several instances within the insurance agency schema, attribute values were derived by matching on the client's name. In step 3 of the application of the SDM (chapter 5) the designer is instructed to list the attributes for each class. Finally, in step 6, the designer is asked to identify attribute interrelationships (inverse or match) and at this time it is realized that attributes that are the subject of a match must have the same attribute name. This discovery required several name changes within the insurance agency schema. Specifically, it was found that a client's name had been represented as Client_name and Name_of_Client in almost equal proportions. Because the name of the client is used extensively for matching, its identifier was standardized as Client_name.

Arguments can be made in favor of standardization if for no other reason than it enhances understanding. However, the designer should be allowed the
freedom of assigning attribute names based upon their role or based upon what is familiar to the end user. No practical solution to this dilemma is suggested other than perhaps including a step 0 to the design steps which would read:

0. Read all of the steps before continuing.

The addition of this new step provides a perfect lead into the discussion of the topic concerning iterative use of the design steps. It is unfortunate but often true that new concepts or procedures are not fully understood until they have been put to use. It was necessary for me to blunder through the design steps several times before fully appreciating what they implied. The most useful lesson learned was to keep subsequent steps in mind while performing each other step. Knowledge of what comes next helps in avoiding certain pitfalls such as the attribute naming problem just discussed. However, knowing the steps that follow at times led to being sidetracked into describing a class in its finest detail which included describing supporting classes in their finest detail. This soon led to utter chaos. The chaos resulted from forgetting plans for the other classes.

In (11) it is suggested that related attributes be grouped together in some type of hierarchy. The example is one of the attribute Size of class SHIP which in turn has attributes Length and Draft. In this highly redundant insurance environment, there are many classes which share common attributes. This was pointed out in Chapter 5. The designers chose to group these common attributes into classes of their own and reference them as a value class. A problem arises when one wishes to reference one of the attributes that is tucked away as an attribute of a value class. To one unfamiliar with SDM or to a casual user a reference to one of these attributes is unclear and confusing. Such a situation prompted the modifications to the match
derivation discussed in Chapter 5 and demonstrated in figure 5-6. Two modifications were suggested to remedy the situation. One would apply the use of mapping, and the second would provide a means for identifying the class to which the match attribute belonged (figure 7-1). Suggested DbDL changes appear in figure 7-2.

Spouse_name
  value class: PERSON_NAME
  match: Client_spouse_info.Name of CLIENT on Client_name

or

Spouse_name
  value class: PERSON_NAMES
  match: Name SPOUSE_INFO on Client_name of CLIENT

Figure 7-1: Alternative descriptions of a match descriptor using proposed syntax extensions illustrated in Figure 7-1.

match: Mapping of CLASS_NAME on ATTRIBUTE_NAME

or

match: ATTRIBUTE_NAME of CLASS_NAME on ATTRIBUTE_NAME of CLASS_NAME.

Figure 7-2 Proposed modifications to SDM DbDL to accommodate matching within attribute classes.

The reader will notice that the class targetted as containing the desired information is different depending upon the method of representation chosen. The class CLIENT is the target in the first example which uses mapping to ultimately identify the attribute containing the value desired. The class SPOUSE_INFO is the target in the second example which requires that the location (class containing the attribute) of the match attribute be specified.

Although the introduction of attribute classes has caused some DbDL
problems, it is not without precedent (5). Attribute classes are extremely useful conceptual descriptors. How would the notion of multiple dependents (Client_dependent_info of CLIENT Appendix D), all with their own unique attribute values, be modeled without the use of the attribute class DEPENDENT_INFO? Or, how would all the information pertaining to each insurance policy, annuity or IRA owned by each client be represented conceptually without the use of their corresponding attribute classes? Other examples include Client_annuities of INSURANCE_APPLICATION_INFO_SUMMARY; Loan_info of POLICY_RECORD; Client_policy_info of CLIENT_FILE_CARD; Client_dependent_info of Client; Employee_dependent_info of EMPLOYEE; etc.

In summary, the SDM is an extremely useful and semantically rich modeling tool. However, even though its designers attempted to keep it as simple as possible with a minimum amount of syntax, the SDM is unwieldy and complicated. The lack of strictly enforceable design steps allows a designer to become easily sidetracked and muddled in detail. The only recourse one has is to justify the creation of each class and attribute and detail their intended use within the description of each. For those who have already been concerned with implementation details in other projects, casual use of the SDM can lead to confusion between conceptual modeling and implementation considerations. It is very easy to project past conceptual considerations and allow implementation issues to restrict and limit the semantic potential of the SDM.

It is already apparent that extensions to the SDM will be necessary to accommodate unique situations. The addition of new constructs should be closely scrutinized. It is easy to envision adding DdDL extensions ad infinitum in lieu of solving complicated problems or special case descriptions with the established DdDL.
Future work

Although the SDM is by comparison simpler to understand and use than Sowa's conceptual graphs, it is apparent from the schema developed in this report that a thorough description of an application environment can become quite complicated. If the SDM is to blossom as a usable conceptual modeling tool, it must be incorporated into an interactive computer program. Arguments supporting this premise have already been presented in this chapter. Ideally an interactive SDM system would contain the following features:

--- A menu driven mode of operation which would provide each alternative available at each stage of a class description. As information is gathered it will be added to a growing knowledge base which will be used to enforce the DbDL syntax.

--- The system would not require that a class be described in its entirety at any one time. The designer may choose to simply enter all class names; or may choose to describe each class in its entirety or any level in between. The system would keep track of the stage of development of each class and could be queried as to the status of each class description or provide detailed reports similar to the "who references what" reports available from some data dictionary systems.

--- The system should provide some form of graphic representation of the developing schema. A hierarchical or network representation might be the most appropriate.

--- For inexperienced users or students the system could provide examples from an existing SDM schema to illustrate the use of specific parts of the DbDL syntax.

--- It should provide a means for defining and incorporating extensions to the DbDL.

--- An option that would allow generation of logical schema of your choice
from the SDM schema would be nice, but realistically the break down of
the SDM schema into the beginnings of a relational model would be most
likely within the scope of such a system.
Appendix A
Syntax of the SDM Data Definition Language

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 "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_FEATURE;INTERCLASS_CONNECTION]]
  [MEMBER_ATTRIBUTE]
  [CLASS_ATTRIBUTE]

CLASS_NAME <-
  #string of capitals possibly including special characters#

CLASS_DESCRIPTION <-
  #string#

BASE_CLASS_FEATURE <-
  [[duplicates allowed; duplicates not allowed]]
  [<<IDENTIFIERS>>]

IDENTIFIERS <-
  [ATTRIBUTE_NAME;ATTRIBUTE_NAME+IDENTIFIERS]

MEMBER_ATTRIBUTES <-
  member attributes:
    <<MEMBER_ATTRIBUTE>>

CLASS_ATTRIBUTES
  class attributes:
    <<CLASS_ATTRIBUTES>>

INTERCLASS CONNECTION <-
  [SUB_CLASS; 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 SCALAR_COMPARATOR [CONSTANT;MAPPING];
  MAPPING SET_COMPARATOR [CONSTANT; CLASS_NAME; MAPPING]]

MAPPING <-
  [ATTRIBUTE_NAME;MAPPING.ATTRIBUTE_NAME]

  [EQUAL_COMPARATOR; > ; >= ; < ; <= ]
EQUAL_COMPARATOR <-
[ - ; <> ]

SET_COMPARATOR <-
[ is {properly} contained in; {properly} contains ]

CONSTANT <-
*a string or number constant*

FORMAT <-
*a name class definition pattern*

MEMBER_ATTRIBUTE <-
<ATTRIBUTE_DESCRIPTION>
{ATTRIBUTE_DESCRIPTION}
value class: CLASS_NAME
{attribute: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_DESCRIPTION>
{ATTRIBUTE_DESCRIPTION}
value class: CLASS_NAME
{derivation:MEMBER_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 <-
[INTERATTRIBUTE_DERIVATION; MEMBER_SPECIFIED_DERIVATION]

CLASS_ATTRIBUTE_DERIVATION <-
[INTERATTRIBUTE_DERIVATION; CLASS_SPECIFIED_DERIVATION]

INTERATTRIBUTE_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_SPECIFIED_DERIVATION <-
[order by [increasing; decreasing] <MAPPING>
{within <MAPPING>};
if in CLASS_NAME;
[up to CONSTANT; all] levels of values of ATTRIBUTE_NAME;
contents]

CLASS_SPECIFIED_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
The Semantic Database Model DbDL Syntax represented in Warnier-like diagrams.

```
<CLASS_NAME> <- #string of capitals possibly including special characters#

description: CLASS_DESCRIPTION) <- #string#

[BASE_CLASS_FEATURE; <-
  duplicates allowed
  duplicates not allowed

  [INTERFACE; <-
    (<<IDENTIFIERS>>) <-
      [ATTRIBUTE_NAME; <- #string of lowercase letters beginning with
                          a capital and possibly including special characters#
        ATTRIBUTE_NAME+IDENTIFIERS]

  SUB_CLASS
  GROUPING_CLASS

(MEMBER_ATTRIBUTE)

(CLASS_ATTRIBUTE)
```
SUBCLASS ← subclass of CLASS_NAME
  where SUBCLASS_PREDICATE ← SUBCLASS_PREDICATE ←

  MAPPING SCALAR_COMPARATOR CONSTANT;
  MAPPING SCALAR_COMPARATOR MAPPING;
  [SIMPLE_PREDICATE; ← MAPPING SET_COMPARATOR CONSTANT;
  [ATTRIBUTE_PREDICATE; ←]
  MAPPING SET_COMPARATOR CLASS_NAME;
  MAPPING SET_COMPARATOR MAPPING]

  (ATTRIBUTE_PREDICATE);

  not ATTRIBUTE_PREDICATE;

  ATTRIBUTE_PREDICATE and ATTRIBUTE_PREDICATE;

  ATTRIBUTE_PREDICATE or 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] ← *a name class definition pattern*
MAPPING <- [ATTRIBUTE_NAME;
  MAPPING.ATTRIBUTE_NAME]

SCALAR_COMPARATOR <- [EQUAL_COMPARATOR; <- [>; <=]; >; <=]

CONSTANT <- *a string or number constant*

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

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]
<ATTRIBUTE_NAME> \leftarrow \text{See \texttt{SUBCLASS}}

\begin{itemize}
\item \texttt{[ATTRIBUTE_DESCRIPTION]} \leftarrow \"\text{#string}\"
\item \text{value class: CLASS_NAME}
\item \text{[inverse:ATTRIBUTE_NAME]}
\item \text{[[match:ATTRIBUTE_NAME of CLASS_NAME on ATTRIBUTE_NAME]]}
\end{itemize}

\texttt{MEMBER\_ATTRIBUTE} \leftarrow \text{derivation:MEMBER\_ATTRIBUTE\_DERIVATION} \leftarrow

\begin{itemize}
\item \texttt{[single valued;}
\item \texttt{multivalued [with size between CONSTANT and CONSTANT]}
\item \texttt{[may not be null]}
\item \texttt{[not changeable]}
\item \texttt{[exhausts value class]}
\item \texttt{[no overlap in values]}
\end{itemize}

\begin{itemize}
\item \texttt{[same as MAPPING; \leftarrow \text{See SUBCLASS for MAPPING}}
\item \texttt{subvalue of MAPPING where is in CLASS_NAME;}
\item \texttt{subvalue of MAPPING where is in ATTRIBUTE\_PREDICATE;}
\item \texttt{where [is in MAPPING and is in MAPPING;}
\item \texttt{where is in MAPPING or is not in MAPPING;}
\item \texttt{where is in MAPPING and is not in MAPPING]}
\end{itemize}

\begin{itemize}
\item \texttt{[MAPPING;}
\item \texttt{=MAPPING \_ EXPRESSION; \leftarrow \{MAPPING};
\item \texttt{MAPPING \_ NUMBER\_OPERATOR \leftarrow \#;}
\item \texttt{MAPPING]}
\end{itemize}

\begin{itemize}
\item \texttt{maximum of MAPPING;}
\item \texttt{minimum of MAPPING;}
\item \texttt{average of MAPPING;}
\item \texttt{sum of MAPPING]}
\item \texttt{number of \{unique\} members in MAPPING;}
\end{itemize}

\begin{itemize}
\item \texttt{[order by increasing \{MAPPING\} \{within \{MAPPING\};}
\item \texttt{order by decreasing \{MAPPING\} \{within \{MAPPING\};}
\item \texttt{MEMBER\_SPECIFIED\_DERIVATION] \leftarrow \text{if in CLASS_NAME;}
\item \texttt{up to CONSTANT levels of value of ATTRIBUTE\_NAME]}
\item \texttt{all levels of value of ATTRIBUTE\_NAME;}
\item \texttt{[contents]}
\end{itemize}
<ATTRIBUTE_NAME> ← see subclass

{ATTRIBUTE_DESCRIPTION} ← "string"

value class: CLASS_NAME

CLASS_ATTRIBUTE ← [derivation:CLASS_ATTRIBUTE_DERIVATION] ←

{single valued

multivalued [with size between
CONStANT and CONSTANT]

{may not be null

{not changeable}
Appendix C
SDM schema after step 3 Chapter 5.

ANNUAL_REVIEW_SCHEDULE
	description: An annual review is conducted for each client once a year. The next review is scheduled 1 year from the last review. A notice is posted in the scheduler one week prior to the review date.
	member attributes:
	Name_of_agent
	Client's_name
	Date_of_appointment
	Time_of_appointment

ANNUAL_STATE_OF_BUSINESS
	description: An annual report of all debits and credits.

APS_FOLLOW-UP
	description: A follow-up to insure that a client's medical papers are received by the appropriate insurance company when new policies are purchased.
	member attributes:
	Name_of_agent (6)
	Client_name (6)
	Policy_number (6)
	Policy_type (6)
	Company_name (6)
	Name_of_doctor
	Doctors_phone
	Date_of_call
	Time_of_call

class attributes:
	Call_number

CASH_FLOW
	description: That data stored in the database that would be of use when filling out a client's cash flow statement.
	member attributes:
	Client_name
	Client_address
	Client's_total_yearly_insurance_premiums

CLIENT
	description: individuals who purchase products from the enterprise.
	member attributes:
	Name
	Home_address (1)
	Social_security_number (1)
	Phone_number (1)
	Date_of_birth (1)
	Spouse's_name (2)
	Spouse's_address (2)
	Spouse's_data_of_birth (2)
	Spouse's_phone_number (2)
	Dependent's_name (3)
	Dependent's_date_of_birth (3)
	Occupation (4)
	Business_phone (4)
	Business_address (4)
	Miscellaneous_information
	Insurance_purchased
	Annuities_owned
	IRA's_owned

CLIENT_ASSET_SHEET
	description: That data stored in the database that would be of use when filling out a client's asset sheet.
	member attributes:
	Client_name
	Client_address
	Spouse_name
	Client's_total_life_insurance
	Spouse's_total_life_insurance

CLIENT_COURTESY_CARD
	description: Courtesy cards are sent to clients on special occasions, e.g., birthdays.
	member attributes:
	Name_of_agent
	Client_name
	Client_address
	Occasion

117
CLIENT_FILE_CARD

description: a summary of the
client attributes, history of
client contacts and a summary of
POLICY_RECORD.

member attributes:
Name
Occupation (4)
Home_address (1)
Home_phone (1)
Business_address (4)
Business_phone (4)
Referred_to_agency_by
Reference_phone
Miscellaneous_information
Approximate_income
Spouse's_name (2)
Spouse's_date_of_birth (2)
Children__names (3)
Client's_date_of_birth (1)
Children's_dates_of_birth (3)
Insurance_owned_.dollar_value
Best_time_to_contact_at_home
Best_time_to_contact_at_work
Contact_record
Company_name (5)
Policy_number (5)
Face_amount (5)
Type (5)
Date_of_issue (5)
Age_of_issue (5)
Gross_yearly_premium
WP_rider (5)
Dividend_option (5)
Primary_beneficiary (5)
Mode_of_premium_payment
Premium_amount
Date_premium_due

CLIENT'S_INSURANCE_AND_ANNUITIES
_IN_FORCE

description: A summary of all
insurance and annuities that a
client has with the agency.

member attributes:
Company_name
Amount
Year_issued
Accidental_death

COMPANIES

description: the companies whose
products are sold by the enterprise.

member attributes:
Company_type
Company_name
Company_address
Company_phone

CREDITS

description: All funds coming into
the agency.

member attributes:
Date
Amount
Source
Person_responsible

DEBITS

description: All funds going out
of the agency.

member attributes:
Date
Amount
Description_of_expenditure
To_whom_charged

CLIENT_FOLLOW-UP

description: an event, required to
insure that all materials
necessary for the purchase of
insurance have been received by
the insurance company.

member attribute:
Name_of_agent (6)
Client_name (6)
Policy_number (6)
Policy_type (6)
Company_name (6)
Time_of_call
Date_of_call
Next_action_required
Date_of_next_action

118
EMPLOYEE
description: all people who work for the enterprise.
member attributes:
  Name
  Home_address (1)
  Social_security_number (1)
  Phone_number (1)
  Date_of_birth (1)
  Spouse's_name (2)
  Spouse's_address (2)
  Spouse's_phone_number (2)
  Spouse's_date_of_birth (2)
  Dependents_name (3)
  Dependents_data_of_birth (3)
  Date_joined_company
  Status_of
  Pay
  Seniority
class attributes:
  Number_of_employees

EVENT
description: a base class needed in order to define the various events that must be scheduled.
member attributes:
  Event_type

INSURANCE_APPLICATION_INFO_SUMMARY
description: That data stored in the database that would be of use when filling out an insurance application form for a current client.
member attributes:
  Client_name
  Client_address
  Spouse's_name
  Spouse's_date_of_birth
  Dependent's_name
  Dependent's_date_of_birth
  Spouse's_amount_insurance
  Client's_insurance_and_annuities_in_force

FORM_SUMMARIES
description: a base class needed in order to define the various form summaries as subclasses.
member attributes:
  Form_number

MONTHLY_CASH_FLOW
description: A monthly report of debits and credits.
member attributes:
  Debits
  Credits

OTHER_APPOINTMENTS
description: Appointments other then those that are normally scheduled.
member attributes:
  Name_of_agent
  Client_name
  Date_of_appointment
  Time_of_appointment
  Description_of_appointment

OUT_OF_TOWN_FOLLOW-UP
description: The agent is required to check with the insurance company about the receipt of initial premium payments by new out of town clients.
member attributes:
  Name_of_agent (6)
  Client_name (6)
  Policy_number (6)
  Policy_type (6)
  Company_name (6)
  Date_of_call

POLICY_RECORD
description: a collection of specifications about insurance policies owned by each client.
member attributes:
  Company (5)
  Policy_number (5)
  Face_amount (5)
  Type (5)
  Date_of.issue (5)
  Age_of.issue (5)
  Gross_yearly_premium
  WP_rider (5)
  Add_rider
  Cpd_rider
  Apl_rider
  Dividend_option (5)
  Loans
  Loan_interest
  Cash_value
  Primary_beneficiary (5)
  Contingent_beneficiary
PRO_FORMA_CASH_FLOW_STATEMENT

description: That data stored in the database that would be of use when filling out a client's pro forma cash flow statement.

member attributes:
Client_name
Client_address
Client's_total_yearly_insurance_premiums

PRODUCT_LICENSE

description: agents must be possess a license for each product he sells.

member attributes:
License_type

PRODUCTS

description: the products sold by the enterprise.

member attributes:
Product_type

PROSPECT_FOLLOW_UP

description: A follow-up to the prospect letter is scheduled 3 days after the prospect letter is mailed.

member attributes:
Name_of_agent
Name_of_prospect
Address_of_prospect
Prospect_phone
Contact_made

PROSPECT_LETTER

description: Information concerning contact with a prospective client.

member attributes:
Name_of_agent
Name_of_prospect
Address_of_prospect
Type_of_product
Date_mailed

REINSTALLMENT_FOLLOW_UP

description: Calls to insurance company to ascertain whether a client has paid his premium.

member attributes:
Name_of_agent (6)
Client_name (6)
Policy_number (6)
Policy_type (6)
Company_name (6)
Date_of_call

class attributes
Number-of-call

REPLACEMENT_FOLLOW_UP

description: In the event that a policy must be replaced the agent is to check with the insurance company concerning issuance of the new policy.

member attributes:
Name_of_agent (6)
Client_name (6)
Policy_number (6)
Policy_type (6)
Company_name (6)
Date_of_call

STATEMENT_OF_FINANCIAL_POSITION

description: That data stored in the database that would be of use when filling out a client's statement of financial position.

member attributes:
Client_name
Client_address
Total_life_insurance_cash_value
Address
PERSONAL_INFO
description: The personal information relevant to each person, both client and employee.
member attributes:
Home_address
Social_security_number
Phone_number
Date_of_birth

SPOUSE_INFO
description: That information relevant to a client's or employee's spouse.
member attributes:
Name
Home_address
Phone_number
Date_of_birth

DEPENDENT_INFO
description: That information relevant to a client's or employee's dependents.
member attributes:
Name
Date_of_birth

CLIENT_BUSINESS_INFO
description: That information relevant to a client's job.
member attributes:
Occupation
Phone_number

POLICY_INFO
description: That information reference most as a group when dealing with client insurance policies.
member attributes:
Company
Policy_number
Type
Face_amount
Date_of_issue
Age_of_issue
Primary_beneficiary
WP_rider
Dividend_option

POLICY_INFO_FOR_FOLLOW-UP
description: That information referenced most as a group when performing follow-ups.
member attributes:
Name_of_agent
Name_of_client
Policy_number
Policy_type
Company
ADDRESS
description: Person's home address.

House_Apt#
  value class: HOUSE_APT
Street_Route
  value class: STREET_ROUTE
State
  value class: STATES
Zip_code
  value class: ZIP_CODES

AGE
description: The values which may be used to describe a person's age.
member attributes:
  Months
    value class: MONTH_VALUES
  Years
    value class: YEAR_VALUES

AGENT
description: The owner and primary agent of the enterprise.
interclass connection: subclass of EMPLOYEE where Status_of="Agent"
member attributes:
  Agent_particulars
    value class: AGENT_INFO
  Agent_commission
    description: The agent receives a commission on all products sold by
                 the agency. The rate varies with type and age of policy. The
                 details of which are beyond the scope of this report.
    value class: DOLLARS

AGENT_INFO
description: Attributes that the agent and subagents have in common. Used as
              a value class for Agent_particulars of AGENT and Subagent_particulars of
              SUBAGENT.
member attributes:
  Clients_of
    value class: CLIENT
    inverse: Agent_of_record
    multivalued
    exhausts value class
    no overlap in values
  Licensed_with
    value class: COMPANIES
    multivalued
  Number_of_clients
    value class: INTEGERS
    derivation: number of unique members in Client_of
ANNUAL_REVIEW_SCHEDULE

description: An annual review is conducted for each client once a year. The next review is scheduled 1 year from the last review. A notice is posted in the schedule one week prior to the review date. Some of the attribute values may be derived from attributes of CLIENT (match on client name).

member attributes:

Name_of_agent
value class: PERSON_NAMES
match: Agent_of_record of CLIENT on Client_name

Client_name
value class: PERSON_NAMES

Date_to_schedule_review_appointment
value class: DATES
derivation: Date_of_last_review + 51 weeks
**** Note: This is not part of the SDM syntax. ****

Date_of_last_review
value class: DATES
derivation: Date client purchased first policy
if new client
or Date_of_appointment
if client has already had at least 1 annual review
and SYSTEM_DATE > Date_of_appointment
**** Note: This is not part of the SDM syntax. ****

Date_of_appointment
value class: DATES

Time_of_appointment
value class: TIME

identifiers: Client_name + Date_of_appointment

ANNUAL_STATE_OF_BUSINESS

description: An annual report of all debits and credits. The "where specified" clause would identify the subset of the members of DEBITS_CREDITS that would belong to this class, e.g., those from the last year.

interclass connection: grouping of DEBITS_CREDITS where specified

member attributes:

Last_year's_debits
value class: DOLLARS
derivation: sum of Debit_amount

Last_year's_credits
value class: DOLLARS
derivation: sum of Credit_amount

Annual_report_date
value class: DATES

identifiers: Annual_report_date

ANNUITIES_REGISTRATION

description: The values an annuity identifier may have.
interclass connection: subclass of STRINGS where specified
ANNUITY_INFO

description: Information about the annuities owned by a client.
member attributes:
  Company
    value class: COMPANY_NAMES
  Annuity_identification
    value class: ANNUITY_REGISTRATION
  Annuity_value
    value class: DOLLARS
class attributes:
  Total_value_of_annuities
    value class: DOLLARS
derivation: sum of Annuity_value

APPOINTMENT_DESCRIPTION

description: A description of an appointment that is not covered under
one the other appointment categories.
interclass connection: subclass of STRINGS where specified

APS_ACTIONS

description: The actions that may be taken when dealing with an APS
follow-up.
interclass connection: subclass of STRINGS where specified
APS_FOLLOW-UP

description: A follow-up to insure that a client's medical papers are received by the appropriate insurance company when new policies are purchased. Several attribute values may be derived from attribute values of POLICY_RECORD (match on Client_name or Policy_number).

member attributes:

Name_of_agent
value class: PERSON_NAMES
match: Agent_of_record of CLIENT on Client_name

Client_name
value class: PERSON_NAMES
match: Client_name of POLICY_RECORD on Policy_number

Policy_number
value class: NUMBERS

Policy_type
value class: POLICY_TYPES
match: Type of POLICY_RECORD on Policy_number

Company
value class: COMPANY_NAMES
match: Company of POLICY_RECORD on Policy_number

Name_of_doctor
value class: PERSON_NAMES

Doctors_phone
value class: PHONE_NUMBERS

Date_of_next_action
value class: DATES

derivation: = Date_of_last_action + 5 working days
    if Last_action="received APS" and APS_number < 2
    or = Date_of_last_action + 2 working days
    if Last_action="called doctor's office"
    or = Date_of_last_action + 4 working days
    if Last_action="received APS and APS_number > 1"
    or = Date_of_last_action + 1
    if Contact_completed of DAILY_SCHEDULE="no"

### Note: This is not part of the SDM syntax. ###

Date_of_last_action
value class: DATES

derivation: = Date_of_next_action
    if Date_of_next_action >= SYSTEM_DATE
    or = Date_of_next_action
    if APS_number = "1"

### Note: This is not part of the SDM syntax. ###

Last_action
description: The action taken on last date of contact.
value class: APS_ACTIONS

class attributes:

APS_number
description: the number of APS forms received by the agency.
value class: NUMBERS

identifiers: Client_name + Policy_number

BOOLEAN
description: The values "yes" or "no".
interclass connection: subclass STRINGS where format is "yes" or "no"
CASH_FLOW

description: That data stored in the database that would be of use when filling out a client's cash flow statement. Attribute values may be derived from attribute values of CLIENT and POLICY_RECORD, match on client name.
member attributes:
  Client_name
    value class: PERSON_NAMES
  Client_address
    value class: ADDRESSES
    match: Client_personal_info.Address of CLIENT on Client_name
  Client's_total_yearly_insurance_premiums
    value class: DOLLARS
    match: Total_gross_yearly_premium of POLICY_RECORD on Client_name
identifiers: Client_name

CASH_VALUE_AT_AGE

description: Cash value of policy at specified age.
member attributes:
  Age
    value class: YEAR_VALUE
  Guaranteed_value
    value class: DOLLARS
  Total_value
    value class: DOLLARS

CASH_VALUE_FIRST_DATE

description: Cash value of policy at first specified date.
member attributes:
  Date
    value class: DATES
  Guaranteed_value
    value class: DOLLARS
  Total_value
    value class: DOLLARS

CASH_VALUE_SCHEDULE

description: A record of the cash value of a policy at specific times.
member attributes:
  Value_first_date
    value class: CASH_VALUE_FIRST_DATE
  Value_second_date
    value class: CASH_VALUE_SECOND_DATE
  Value_at_specific_age
    value class: CASH_VALUE_AT_AGE

CASH_VALUE_SECOND_DATE

description: Cash value of policy at second specified date.
member attributes:
  Date
    value class: DATES
  Guaranteed_value
    value class: DOLLARS
  Total_value
    value class: DOLLARS
CLIENT

definition: Individuals who purchase products from the enterprise.
member attributes:
  Client_name
    value class: PERSON_NAMES
  Client_personal_info
    value class: PERSONAL_INFO
  Client_spouse_info
    value class: SPOUSE_INFO
  Client_dependent_info
    value class: DEPENDENT_INFO
    multivalued
  Client_business_info
    value class: CLIENT_BUSINESS_INFO
  Miscellaneous_information
    value class: MISC_INFO
  Annuities_owned
    value class: ANNUITIES_INFO
    multivalued
  IRA's Owned
    value class: IRA_INFO
    multivalued
  Agent_of_record
    value class: AGENT_INFO
    inverse: Client_of
  identifiers: Client_name
CLIENT_ASSET_SHEET

description: That data stored in the database that would be of use when filling out a client's asset sheet.

member attributes:

Client_name
  value class: PERSON_NAMES

Client_address
  value class: ADDRESSES
  match: Client_personal_info.Address of CLIENT on Client_name

Spouse_name
  value class: PERSON_NAMES
  match: Client_spouse_info.Spouse_name of CLIENT Client_name

Client's_total_life_insurance
  value class: DOLLARS
  match: Insurance_owned._dollar_value of CLIENT_FILE_CARD on Client_name

Spouse_total_life_insurance
  value class: DOLLARS
  match: Total_insurance_owned of POLICY_RECORD on Spouse_name

*** Note: The intent was to derive the amount of insurance owned by the spouse by matching her name (Spouse_name) on a Client_name of POLICY_RECORD. This is assuming that the only knowledge of insurance that the agency should be expected to have is that which they have sold. If the spouse has any insurance with the agency, he/she should also be considered a client and thus he/she would have a value for Total_insurance_owned. It is not clear as to the values that are actually being matched. The intent was to match Spouse_name with a Client_name in the class POLICY_RECORD. ***

identifiers: Client_name

CLIENT_BUSINESS_INFO

description: That information relevant to a client's job.

member attributes:

Occupation
  value class: OCCUPATIONS

Phone_number
  value class: PHONE_NUMBERS

Address
  value class: ADDRESSES
CLIENT_COURTESY_CARD

description: Courtesy cards are sent to clients on special occasions, e.g., birthdays. Attribute values may be derived from attribute values of CLIENT, match on client name.

member attributes:
  Name_of_agent
    value class: PERSON_NAMES
  Clients_name
    value class: PERSON_NAMES
  Client_address
    value class: ADDRESSES
      match: Client_personal_info.Address of CLIENT on Client_name
  Client_birthday
    value class: DATES
      match: Client_personal_info.Date_of_birth of PERSONAL on Client_name
  Date_to_send_card
    value class: DATES
      derivation: Client_birthday - 3 working days

identifiers: Client_name + Date_to_send_card
CLIENT_FILE_CARD

description: A summary of the client attributes, history of client contacts and a summary of POLICY_RECORD. Most attributes pertaining to insurance will be derived from attributes of POLICY_RECORD based on a match with Policy_number of CLIENT_FILE_CARD POLICY_INFO. Those attributes pertaining to the client will be derived from attributes of CLIENT based on a match with Client_name.

member attributes:

Client_name
  value class: PERSON_NAMES
Client_personal_info
  value class: PERSONAL_INFO
  match: Client_personal_info of CLIENT on Client_name
Client_spouse_info
  value class: SPouse_INFO
  match: Client_spouse_info of CLIENT on Client_name
Client_dependent_info
  value class: DEPENDENT_INFO
  match: Client_dependent_info of CLIENT on Client_name
Client_business_info
  value class: CLIENT_BUSINESS_INFO
  match: Client_business_info of CLIENT on Client_name
Referrer_to_agency_by
  value class: PERSON_NAMES
Reference_phone
  value class: PHONE_NUMBERS
Miscellaneous_information
  value class: MISCINFO
Approximate_income
  value class: DOLLARS
Insurance_owned, dollar_value
  value class: DOLLARS
  match: Total_insurance_owned of POLICY_RECORD on Client_name
Best_time_to_contact_at_home
  value class: TIMES
Best_time_to_contact_at_work
  value class: TIMES
Client_policy_info
  description: a summary of each policy owned by the client.
  value class: CLIENT_FILE_CARD POLICY_INFO
  multivalued
Contact_record
  value class: APS_FOLLOW-UP, REINSTATEMENT_FOLLOW-UP,
  PROSPECT_FOLLOW-UP, REPLACEMENT_FOLLOW-UP, OTHER APPOINTMENTS
  OUT_OF_TOWN_FOLLOW-UP, ANNUAL REVIEW, CLIENT COURTESY CARD
  match: Event of DAILY SCHEDULE on Client_name (of appropriate class)

*** Note: This is not part of the SDM syntax. ***

identifiers: Client_name

130
CLIENT_FILE_CARD_POLICY_INFO

description: A summary of the policies a client owns. Attribute values may be derived from attribute values of POLICY_RECORD, match on Policy_number.

member attributes:

Policy_number

description: A client will have 1 instance of POLICY_RECORD for each policy owned. The policy numbers of the policies owned by each client can be derived from POLICY_RECORD. A match on the client's name will result in several policy numbers each of which should be used in turn to provide multiple instances of Client_policy_info, an attribute of CLIENT_FILE_CARD whose value class is CLIENT_FILE_CARD_POLICY_INFO.

value class: POLICY_NUMBERS
match: Policy_number of POLICY_RECORD on Client_name
(of CLIENT_FILE_CARD)

Plan

value class: POLICY_TYPES
match: Type of POLICY_RECORD on Policy_number

Company

value class: COMPANY_NAMES
match: Company of POLICY_RECORD on Policy_number

Mode_of_premium_payment

value class: MODE_OF_PAYMENT

Premium_amount

value class: DOLLARS
match: Premium_amount of POLICY_RECORD on Policy_number

Date_premium_due

value class: DATES
match: Date_premium_due of POLICY_RECORD on Policy_number

Face_amount

value class: DOLLARS
match: Face_amount of POLICY_RECORD on Policy_number

Date_of_issue

value class: DATES
match: Date_of_issue of POLICY_RECORD on Policy_number

Age_of_issue

value class: AGE
match: Age_of_issue of POLICY_RECORD on Policy_number

Primary_beneficiary

value class: PERSON_NAMES
match: Primary_beneficiary of POLICY_RECORD on Policy_number

WP_rider

value class: BOOLEAN
match: WP_rider of POLICY_RECORD on Policy_number

Dividend_option

value class: BOOLEAN
match: Dividend_option of POLICY_RECORD on Policy_number
CLIENT'S INSURANCE IN FORCE

description: A summary of all insurance and annuities that a client has with the agency. This information is needed for item 15 of form 1459 (insurance application).

member attributes:
  Company_name
    value class: COMPANY_NAMES
    match: Company of POLICY_RECORD on Client_name (of INSURANCE_APPLICATION_INFO_SUMMARY)
  Amount
    value class: DOLLARS
    match: Face_amount of POLICY_RECORD on Client_name (of INSURANCE_APPLICATION_INFO_SUMMARY)
  Date_issued
    value class: DATES
    match: Date_of_issue of POLICY_RECORD on Client_name (of INSURANCE_APPLICATION_INFO_SUMMARY)
  Accidental_death
    description: If the policy has an accidental death rider then Accidental_death = "yes"
    value class: BOOLEAN
    match: Add_rider of POLICY_RECORD on Client_name (of INSURANCE_APPLICATION_INFO_SUMMARY)

COMPANIES

description: The companies whose products are sold by the enterprise.

member attributes:
  Company_type
    value class: COMPANY_TYPES
  Company_name
  value class: COMPANY_NAMES
  Company_address
  value class: ADDRESSES
  Company_phone
  value class: PHONE_NUMBERS
  Products_offered
  value class: PRODUCT_TYPE
  identifiers: Company_name

COMPANY_NAMES

description: The companies represented by the agency.

interclass connection: subclass of STRINGS where specified

COMPANY_TYPES

description: The companies whose products are sold by the agency.

interclass connection: subclass of STRINGS where specified

derivation: order by Debit_date
DAILY_SCHEDULE

description: A schedule of events (follow-ups, appointments, etc.) that are to occur on the current day.

member attributes:
  Event
    value class: AFS_FOLLOW-UP, REINSTATEMENT_FOLLOW-UP,
    PROSPECT_FOLLOW-UP, REPLACEMENT_FOLLOW-UP, OTHER_APPOINTMENTS
    OUT_OF_TOWN_FOLLOW-UP, ANNUAL_REVIEW, CLIENT_COURTESY_CARD
  derivation: where Date_of_next_action = Today's_date

  *** Note: This is not part of the SDM syntax. ***

  Today's_date
    value class: SYSTEM_DATE

  Contact_completed
    description: After action has been completed Contact_completed = "yes" otherwise Contact_completed = "no"
    value class: BOOLEAN

identifiers: Client_name (of appropriate value class) + Today's_date

DATES

description: calendar dates in the range "1/1/50" to "12/31/99"

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 > 1950 and < 1999
  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

DEBITS

description: All funds going out of the agency.

interclass connection: subclass of DEBITS CREDITS where
  Transaction_type = "debits"

member attributes:
  Debit_date
    value class: DATES

  Debit_amount
    value class: DOLLARS

  Description_of_expenditure
    value class: EXPENDITURE_DESCRIPTION

  To_whom_charged
    value class: PERSON_NAMES

  Order_of_debit
    description: The chronological order of the debits.
    value class: INTEGERS

DEBITS CREDITS

description: A parent class needed in order to accumulate both debits and credits for monthly, quarterly and annual cash flow reports.

member attributes:
  Transaction_type
    value class: TRANSACTION_TYPES
DEPENDENT_INFO
description: Information relevant to a client's or employee's dependents.
member attributes:
   Name
       value class: PERSON_NAMES
   Date_of_birth
       value class: DATES

DOLLARS
description: The form a reference to money may take.
interclass connection: subclass of STRINGS where format is
        "$
        000 < number <= 999
        ",
        000 <= number <= 999
        ",
        000 <= number <= 999
        ",
        00 <= number <= 99
where leading zeros are omitted

EMPLOYEE
description: all people who work for the enterprise.
member attributes:
   Employee_name
       value class: PERSON_NAMES
   Employee_personal_info
       value class: PERSONAL_INFO
   Employee_spouse_info
       value class: SPOUSE_INFO
   Employee_dependent_info
       value class: DEPENDENT_INFO
       multivalued
   Date_joined_company
       value class: DATES
   Status_of
       value class: EMPLOYEE_TYPES
   Pay
       value class: PAY
   Seniority
       value class: INTEGERS
derivation: order by increasing Date_joined_company
class attributes:
   Number_of_employees
       value class: INTEGERS
derivation: number of members in this class.
identifiers: Employee_name

EMPLOYEE_TYPES
description: The types of employees at the agency, values may be "agent",
"subagent" and "office staff".
interclass connection: subclass of STRINGS where specified
EVENT_TYPES
  description: Each event, i.e., appointments follow-ups, etc., is specified as a specific event type.
  interclass connection: subclass of STRINGS where specified

EXPENDITURE_DESCRIPTION
  description: A description of the type of expenditure list as a debit.
  interclass connection: subclass of STRINGS where specified

FIRST_NAMES
  description: The first name of a person.
  interclass connection: subclass of STRINGS where specified

FORM_NUMBERS
  description: Each summary form has a unique number that is used to reference the form.
  interclass connection: subclass of STRINGS where specified

HOUSE_APT
  description: The house number, apartment number, suite number or office number in an address.
  interclass connection: subclass of STRINGS where specified
INSURANCE_APPLICATION_INFO_SUMMARY

description: That data stored in the database that would be of use when
filling out an insurance application form for a current client.

Attribute values may be derived from attribute values of
CLIENT, SPOUSE_INFO, DEPENDENT_INFO, and CLIENT_FILE_CARD, match on
Client_name.

member attributes:

Client_name
  value class: PERSON_NAMES
Client_address
  value class: ADDRESSES
  match: Client_personal_info.Home_address of CLIENT on Client_name
Spouse_name
  value class: PERSON_NAMES
  match: Name_of_spouse of SPOUSE_INFO on Client_name (of CLIENT)
Spouse_date_of_birth
  value class: DATES
  match: Client_spouse_info.Date_of_birth of CLIENT on Client_name
Dependent's_name
  value class: PERSON_NAMES
  match: Name_of_dependent of DEPENDENT_INFO on Client_name
  (of CLIENT)
Dependent's_date_of_birth
  value class: DATES
  match: Client_dependent_info.Date_of_birth of CLIENT on Client_name
Spouse_amount_insurance
  description: If it is assumed that if the spouse is also a client
  then a record of total insurance owned should also be present
  for the spouse. This value is derivable by matching Spouse_name
to Client_name in POLICY_RECORD.
  value class: DOLLARS
  match: Total_insurance_owned of POLICY_RECORD on
  Spouse_name
Client's_insurance
  value class: CLIENT'S_INSURANCE_IN_FORCE
Client_annuities
  value class: ANNUITY_INFO
  match: Annuities_owned of CLIENT on Client_name (of CLIENT)
identifiers: Client_name

INTEREST

description: The value a interest reference may have.
interclass connection: subclass of STRINGS where format is
00<=number<=99
  
"."
00,=number<=99
"%"
IRA_INFO
description: Information about the IRA's owned by a client.
member attributes:
  Company
    value class: COMPANY_NAMES
  IRA_identification
    value class: IRA_REGISTRATION
  IRA_value
    value class: DOLLARS
  class attributes:
    Total_value_of_IRA's
    value class: DOLLARS
    derivation: sum of IRA_value

IRA_REGISTRATION
description: The values an IRA identifier may have.
interclass connection: subclass of STRINGS where specified

LAST_NAMES
description: The last name of a person, it may be one or more names or hyphenated.
interclass connection: subclass of STRINGS where specified

LOAN_INFO
description: Information about individual insurance loans a client has.
  Loan_amount
    value class: DOLLARS
  Loan_interest
    value class: INTEREST

MIDDLE_NAMES
description: The middle name identifiers a person may have. It may be one or more initials or one or more names.
interclass connection: subclass of STRINGS where specified

MISC_INFO
description: Miscellaneous information.
interclass connection: subclass of STRINGS where specified

MODE_OF_PAYMENT
description: The various payment schedules available.
interclass connection: subclass of STRINGS where specified

MONTH_VALUES
description: The values a month may have.
interclass connection: subclass of INTEGERS where format is 1<=number<=12
MONTHLY_CASH_FLOW

description: A monthly report of all debits and credits. The "where specified" clause would identify the subset of the members of DEBITS_CREDITS that would belong to this class, e.g., those from the last month.

interclass connection: grouping of DEBITS_CREDITS where specified

member attributes:
  Last_month's_debits
    value class: DOLLARS
    derivation: sum of Debit_amount
  Last_month's_credits
    value class: DOLLARS
    derivation: sum of Credit_amount
  Monthly_report_date
    value class: DATES
  identifiers: Monthly_report_date

NEXT_ACTION

description: The next action required for a follow-up.

interclass connection: subclass of STRINGS where specified

OCCASIONS

description: A description of the occasion for a client courtesy card.

interclass connection: subclass of STRINGS where specified

OCCUPATIONS

description: Occupations a client may have.

interclass connection: subclass of STRINGS where specified

OFFICE_STAFF

description: The class that identifies the office staff and the office manager.

interclass connection: subclass of EMPLOYEE where
  Status_of = "office worker"

member attributes:
  Is_office_manager
    description: "yes" if the instance is the office manager
    "no" otherwise.
    value class: BOOLEAN
OUT_OF_TOWN_FOLLOW-UP

description: The agent is required to check with the insurance company about the receipt of initial premium payments by new out of town clients.

member attributes:

Name_of_agent
  value class: PERSON_NAMES
  match: Agent_of_record of CLIENT on Client_name

Client_name
  value class: PERSON_NAMES
  match: Client_name of POLICY_RECORD on Policy_number

Policy_number
  value class: NUMBERS

Policy_type
  value class: POLICY_TYPES
  match: Type of POLICY_RECORD on Policy_number

Company
  value class: COMPANY_NAMES
  match: Company of POLICY_RECORD on Policy_number

Date_of_next_action
  value class: DATES
  derivation: = Date_policy_sold + 1/4 days
    if this is the initial contact
    or =Date_of_next_action + 1
    if the person was not contacted on the Date_of_next_action
    this is denoted by a "no" value of Contact_completed of class DAILY_SCHEDULE for this instance

**** Note: This is not part of the SDM syntax. ****

Date_policy_sold
  value class: DATES

identifiers: Client_name + Date_policy_sold

OTHER_APPOINTMENTS

description: Appointments other than those that are normally scheduled, i.e., the follow-ups and annual review.

member attributes:

Name_of_agent
  value class: PERSON_NAMES

Client_name
  value class: PERSON_NAMES

Date_of_appointment
  value class: DATES

Time_of_appointment
  value class: TIMES

Description_of_appointment
  value class: APPOINTMENT_DESCRIPTION

Next_date_of_action
  description: This attribute must be present if OTHER_APPOINTMENTS is to be included in DAILY_SCHEDULE. Membership in DAILY_SCHEDULE is based on a match with Next_date_of_action. Rather than change the name of the attribute Date_of_appointment, which is familiar to all employees, Next_date_of_action was coined as a synonym.
  value class: DATES
  derivation: same as Date_of_appointment

identifiers: Client_name + Date_of_appointment
PAY

description: The monthly pay for each employee. Pay is based on a base salary plus a commission depending on the value Status_of. In general office staff will receive only a base salary and the agent and subagents will receive only a commission. The details of pay are beyond the scope of this report.

PERSON_NAMES

description: The form a person's name may take.

member attributes:
  Title
    value class: TITLES
  First
    value class: FIRST_NAMES
  Middle
    value class: MIDDLE_NAMES
  Last
    value class: LAST_NAMES

PERSONAL_INFO

description: The personal information relevant to each person, both client and employee.

member attributes:
  Home_address
    value class: ADDRESSES
  SSN
    value class: SOCIAL_SECURITY_NUMBER
  Phone_number
    value class: PHONE_NUMBERS
  Date_of_birth
    value class: DATES

PHONE_NUMBERS

description: The form a phone number may take, area code, prefix, number.

interclass connection: subclass of STRINGS where format is

  000<number><999
  "-"  
  00<number><999
  "-"  
  0000<number><9999

POLICY_NUMBERS

description: The values a policy number may have.

interclass connection: subclass of STRINGS where specified
POLICY_RECORD
description: a collection of specifications about insurance policies owned by each client. The POLICY_RECORD class will act as the master record of each policy sold.
member attributes:
Client_name
  value class: PERSON_NAMES
Policy_number
  value class: POLICY_NUMBERS
Type
  value class: POLICY_TYPES
Company
  value class: COMPANY_NAMES
Face_amount
  value class: DOLLARS
Date_of_issue
  value class: DATES
Age_of_issue
  value class: AGE
Primary_beneficiary
  value class: PERSON_NAMES
WP_rider
  value class: BOOLEAN
Dividend_option
  value class: BOOLEAN
Gross_yearly_premium
  value class: DOLLARS
Add_rider
  value class: BOOLEAN
Cpd_rider
  value class: BOOLEAN
Apl_rider
  value class: BOOLEAN
Loans
  value class: LOAN_INFO
Cash_value_record
  value class: CASH_VALUE_SCHEDULE
Contingent_beneficiary
  value class: PERSON_NAMES
Total_insurance_owned
  value class: DOLLARS
derivation: sum of Face_amount
Total_gross_yearly_premiums
  value class: DOLLARS
derivation: sum of Gross_yearly_premium
identifiers: Client_name + Policy_number

POLICY_TYPES
description: The types of policies sold by the agency.
interclass connection: subclass of STRINGS where specified.
PRO_FORMA_CASH_FLOW_STATEMENT

description: That data stored in the database that would be of use when filling out a client's pro forma cash flow statement. Attribute values may be derived from attribute values of POLICY_RECORD, match on client_name.

member attributes:
Client_name
  value class: PERSON_NAMES
Client_address
  value class: ADDRESSES
Client's_total_yearly_insurance_premiums
  value class: DOLLARS
  match: Total_gross_yearly_premiums of POLICY_RECORD on Client_name

identifiers: Client_name

PRODUCT_LICENSE

description: An agent must possess a license for each product he sells.

member attributes:
License_type
  value class: PRODUCT_TYPES

PRODUCT_TYPES

description: The product types sold by the agency.
interclass connection: subclass STRINGS where specified

PRODUCTS

description: the products sold by the enterprise.

member attributes:
Product_type
  value class: PRODUCT_TYPES
PROSPECT_FOLLOW-UP

description: A follow-up to the prospect letter is scheduled 3 days after
the prospect letter is mailed.

member attributes:

Name_of_agent
  value class: PERSON_NAMES
  match: Name_of_agent of PROSPECT_LETTER on Name_of_prospect

Name_of_prospect
  value class: PERSON_NAMES

Address_of_prospect
  value class: ADDRESSES
  match: Address_of_prospect of PROSPECT_LETTER on Name_of_prospect

Prospect_phone
  value class: PHONE_NUMBERS

Type_of_product
  value class: PRODUCT_TYPES
  match: Type_of_product of PROSPECT_LETTER on Name_of_prospect

Contact_made
  description: If contact was not made on the scheduled
date then value="no".
  value class: BOOLEAN

Date_of_contact
  description: An agent is to contact the prospect 3 working days
after the prospect letter has been mailed.
  value class: DATES
  derivation: = Date_letter_mailed + 3 working days
  or = Date_of_contact + 1
  if Contact_completed of DAILY_SCHEDULE="no"

*** Note: This is not part of the SDM syntax. ***

identifiers: Name_of_prospect

PROSPECT_LETTER

description: Information concerning contact with a prospective client.

member attributes:

Name_of_agent
  value class: PERSON_NAMES

Name_of_prospect
  value class: PERSON_NAMES

Address_of_prospect
  value class: ADDRESSES

Type_of_product
  value class: PRODUCT_TYPES

Date_mailed
  value class: DATES

identifiers: Name_of_prospect
QUARTERLY_REPORT

description: A 3 month report of all debits and credits. The "where
specified" clause would identify the subset of the members of
DEBITS_CREDITS that would belong to this class e.g., those from the
last 3 months.

interclass connection: grouping of DEBITS_CREDITS where specified
member attributes:
  Last_3_months_debits
    value class: DOLLARS
derivation: sum of Debit_amount
  Last_3_months_credits
    value class: DOLLARS
derivation: sum of Credit_amount
  Quarterly_report_date
    value class: DATES
identifiers: Quarterly_report_date

REINSTATEMENT_FOLLOW-UP

description: The agency is to call the insurance company to ascertain
whether a client has paid his premium.

member attributes:
  Name_of_agent
    value class: PERSON_NAMES
    match: Agent_of_record of CLIENT on Client_name
  Client_name
    value class: PERSON NAMES
    match: Client_name of POLICY_RECORD on Policy_number
  Policy_number
    value class: NUMBERS
  Policy_type
    value class: POLICY_TYPES
    match: Type of POLICY_RECORD on Policy_number
  Company
    value class: COMPANY NAMES
    match: Company of POLICY_RECORD on Policy_number
  Date_of_next_action
    value class: DATES
    derivation: = Date_notified_of_delinquency + 14
      if this is the first attempt
      or Date_of_next_action + 1
      if client was not notified on the initial date, denoted
      by a "no" value of Contact_completed of class DAILY_SCHEDULE
      for this instance
      *** Note: This is not part of the SDM syntax. ***
  Date_notified_of_delinquency
    value class: Dates
identifiers: Client_name

144
REPLACEMENT_FOLLOW-UP

description: In the event that a policy must be replaced, the agent is to check with the insurance company concerning issuance of the new policy.

member attributes:

Name_of_agent
value class: PERSON_NAMES
match: Agent_of_record of CLIENT on Client_name

Client_name
value class: PERSON_NAMES
match: Client_name of POLICY_RECORD on Policy_number

Policy_number
value class: NUMBERS

Policy_type
value class: POLICY_TYPES
match: Type of POLICY_RECORD on Policy_number

Company
value class: COMPANY_NAMES
match: Company of POLICY_RECORD on Policy_number

Date_of_next_action
value class: DATES

 derivation: =Date_replacement_policy_sold + 14
    if this is the initial date scheduled
    or Date_of_next_action + 1
    if action was not completed on date specified this is denoted
    by a "no" value of Contact_completed of class DAILY_SCHEDULE
    for this instance

*** Note: This is not part of the SDM syntax. ***

identifiers: Client_name


SOCIAL_SECURITY_NUMBER
interclass connection: subclass of STRINGS where format is
000<=>number<=999
"-

00<=>number<=99
"-

0000<=>number<=9999


SOURCE

description: A description of the source of a credit.
interclass connection: subclass of STRINGS where specified


SPouse_INFO

description: That information relevant to a client's or employee's spouse.

member attributes:

Name
value class: PERSON_NAMES

Home_address
value class: ADDRESSES

Phone_number
value class: PHONE_NUMBERS

Date_of_birth
value class: DATES
STATEMENT_OF_FINANCIAL_POSITION
   description: That data stored in the database that would be of use when filling out a client's statement of financial position.
   member attributes:
      Client_name
         value class: PERSON_NAMES
      Client_address
         value class: ADDRESSES
      Total_life_insurance_cash_value
         value class: DOLLARS
   identifiers: Client_name

STATES
   description: Postal service accepted state identifiers.
   interclass connection: subclass of STRINGS where specified

STREET_ROUTE
   description: The Street or route in an address.
   interclass connection: subclass of STRINGS where specified

SUBAGENT
   description: The persons who work for the agency and are supervised by the Agent.
   interclass connection: subclass of EMPLOYEE where Status_of="Subagent"
   member attributes:
      Subagent_particulars
         value class: AGENT_INFO
      Subagent_commission
         description: Subagents receive an initial commission and residual commission based on the type and age of the product and the company that sells the product. The residual commission is based on the client's payment. The model of this procedure is beyond the scope of this report.
         value class: DOLLARS

SYSTEM_DATE
   description: A new primitive which refers to the current date.
   *** Note: This is not part of the SDM syntax. ***

TIME
   description: The time record using A.M.-P.M. designators.
   interclass connection: subclass STRINGS where format is
      hour: number where > 1 and < 12
      "":""
      minutes: number where >= 00 and < 60
      "":""
      "A.M." or "P.M."

TITLES
   description: The title a person may have preceding his name, e.g., Dr., Mr., Ms., Mrs.
   interclass connection: subclass of STRINGS where specified
TRANSACTION_TYPES
   description: Transaction types are debits or credits.
   interclass connection: subclass of STRINGS where specified

YEAR_VALUE
   description: The value a year may have.
   interclass connection: subclass of INTEGER where format is
   \( 1 \leq \text{number} \leq 130 \)

ZIP CODES
   description: The accepted postal service zip codes.
   interclass connection: subclass of STRINGS where specified
## Appendix E

### Insurance Company Business Forms

### Client File Card

<table>
<thead>
<tr>
<th>NAME</th>
<th>BIRTHDAYS</th>
<th>INSURANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Month</td>
<td>Day</td>
</tr>
<tr>
<td>Residence</td>
<td>PHONE</td>
<td></td>
</tr>
<tr>
<td>OCCUPATION</td>
<td>WIFE'S NAME</td>
<td></td>
</tr>
<tr>
<td>FIRM ADDRESS</td>
<td>PHONE</td>
<td>CHILDREN</td>
</tr>
<tr>
<td>REFERRED BY</td>
<td>PHONE</td>
<td></td>
</tr>
<tr>
<td>Club, Church, Frat., Hobbies</td>
<td>Approx. Income</td>
<td></td>
</tr>
<tr>
<td>BEST TIME TO SEE:</td>
<td>Day</td>
<td></td>
</tr>
<tr>
<td>At Work</td>
<td>Daytime</td>
<td></td>
</tr>
<tr>
<td>At Home</td>
<td>Evening</td>
<td></td>
</tr>
<tr>
<td>CALL DATE</td>
<td>NEED</td>
<td>RESULTS</td>
</tr>
<tr>
<td>OTHER INFORMATION</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### PRESENT LIFE INSURANCE

<table>
<thead>
<tr>
<th>Company</th>
<th>Date Issued</th>
<th>Plan</th>
<th>Amount</th>
<th>Benefits</th>
<th>Premium</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Lafayette Life Insurance Company

Form 1144

148
Insurance Application

Part 1

1. PRINT

   FULL
   NAME
   □ Male □ Married □ Widowed □ Separated
   □ Female □ Single □ Divorced

2. State and Date of Birth: State Month Day Year


4. ADDRESS for Premium Notices:
   Street, Number:
   City State Zip (if needed)

5. POLICYOWNER to be:
   □ Proposed Insured
   □ Other
   Name of Owner Age
   Relationship to Proposed Insured
   Social Security or Tax I.D. No.
   Contingent Owner (if any)

6. Will any existing life or annuity contract be lapsed, surrendered, borrowed against, or changed if the proposed policy is issued or have any of these transactions been completed recently? (If "Yes" furnish name of company being replaced)
   □ Yes □ No

7. OCCUPATIONS: (include Spouse's if benefit applied for)
   How long so Employed?
   Years Months

Home Office Corrections and Additions (Do not write in this space)

"NOT for use in Pa., Md., or W.V.

8. Plan (See Rate Book B-5)

8.a. Amount of Insurance
   Policy ____________________________ $ ____________________________
   Rider ____________________________ $ ____________________________

  for Universal life policies, select plan and option below.
   □ Adj. Whole Life □ Life
   Key UL-83 □ C.V. - included
   □ Value Life - RX-83 □ C.V. - Excluded

9. ADDITIONAL BENEFITS (Check each benefit desired)
   □ Waiver of Premium □ GPO $ ____________
   □ Accidental Death $ ____________
   □ Payor Benefit
   □ Children's Rider $ ____________

   BENEFITS AVAILABLE ONLY WITH UNIVERSAL LIFE
   □ Waiver of Monthly Premiums □ Cost of Living
   □ Adjustable Rider
   □ $ ____________
   □ GIO $ ____________

9.a. DIVIDEND OPTION
   □ Add to Cash Value (Un. Life only)
   □ Paid-Up Additions Accumulated at Interest
   □ Paid in Cash Reduce Premiums

10. PREMIUM PAYMENT
    (The amount of CASH Must agree with Conditional Receipt)
    Planned Periodic Premium $ ____________
    □ Annual □ Semi. □ Quarterly □ PAC □ OA □ SD

10a. Premium Loan provision will be in effect in all ordinary life policies unless "No" is checked.

11. ANNUITIES:
    □ Flex Prem Annuity at Retirement Age ________
    □ Single Premium Deferred

12. LIFE INSURANCE & ANNUITIES IN FORCE:
    Company ________ Amount ________ Year ________ Accidental Death ________

13. BENEFICIARY
    Print full given names Age Relationship
    Primary: ________ ________ ________ ________ ________
    ________ ________ ________ ________ ________
    ________ ________ ________ ________ ________
    Contingent ________ ________ ________ ________ ________
    ________ ________ ________ ________ ________
    □ Lawful children of the insured (including any named above)
    □ Children born of or legally adopted to the marriage of the insured and primary beneficiary (including any named above). Check only if spouse of insured is named as primary beneficiary.

14. Additional Information and Special Requests

Form 1459 - A (Continued on Reverse Side) 5/84
PART A — AVIATION ACTIVITIES — COMPLETE IF NO. 16e, ANSWERED YES

1. Indicate your current, past or intended flying status:
   □ Pilot  □ Student Pilot  □ Crew member or observer  □ Other

2. Do you fly in any military or military reserve capacity or intend to do so?
   □ Yes  □ No
   If so, state capacity:

3. Total hours flown as pilot?

4. Date of last flight?

5. Types of certificates held and issue dates?

6. Do you have an instrument flying rating?
   □ Yes  □ No

7. Have you ever been in an aircraft accident or been grounded for violation of regulations?
   □ Yes  □ No
   If "Yes," explain.

8. Types of flying you do or intend to do:
   □ Pleasure, personal business  □ Scheduled airline
   □ Non-scheduled carrier  □ Business planes
   □ Charter  □ Flight instruction  □ Test-flight production
   □ Test-experimental  □ Dusting, seeding, spraying
   □ Other:

9. Extent of past or anticipated flying by type of flying indicated in Question 8.
   Hours of Flying Time
   Type of Flying  Last 12 Mo.  1 to 2 Yrs. Ago  Next 12 Mo.
   Military Pilot or Crew

10. Principal types of aircraft flown?

11. Should aviation coverage be issued for extra premium, if required?
   □ Yes  □ No

PART B — MILITARY STATUS — COMPLETE IF NO. 16e, ANSWERED YES

1. Are you in the Armed Forces, National Guard or any Reserve Unit?
   □ Yes  □ No
   If "Yes," complete the following:
   a. Branch:  
   b. Pay Grade:  
   c. Job classification:  

2. Current address if different from face of application.

OTHER PERSONS PROPOSED FOR INSURANCE

15. Names of Others to be Insured (Spouse or Children)

<table>
<thead>
<tr>
<th>Relationship to Proposed Insured</th>
<th>Date of Birth</th>
<th>State</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
<th>Amount of Life Insurance in Force</th>
</tr>
</thead>
</table>

QUESTIONS 16 - 22. MUST BE COMPLETED IN ALL CASES

16. Has any person proposed for insurance: (Give DETAILS IN NO. 28 TO ANY "YES" ANSWER.)
   □ Yes  □ No
   a. ever been declined, postponed, excluded or rated for Life or Health insurance or had reinstatement refused?
   □     □
   b. applied for other Life insurance within past 6 months?
   □     □
   c. made or contemplate making any flight as a pilot or crew member of any aircraft? (If "Yes," complete A, above)
   □     □
   d. engaged in, or contemplate engaging in, parachuting, hang gliding, vehicle racing, skin or scuba diving? (Complete special questionnaire)
   □     □
   e. joined any Armed Forces or Advanced ROTC? (If "Yes," complete B, above)
   □     □
   f. Disease or disorder of the stomach, intestine, bowel, rectum, liver or gall bladder?
   □     □
   g. Disease or disorder of the brain or nervous system, fainting spells, epilepsy, convulsion or paralysis?
   □     □
   h. Disease of the urinary system including nephritis, kidney stone, disease of the kidneys, bladder, or prostate?
   □     □
   i. Sugar, albumin, or blood in the urine?
   □     □
   j. Disease or disorder of bone, joints, muscle, back, spine, rheumatism, arthritis, gout, loss of limb or deformity?
   □     □
   k. Any disease or disorder of the eyes, ears, nose or throat?
   □     □
   l. Cancer, tumor, venereal disease, diabetes, leukemia, disorder of glands, blood, breast, or reproductive organs?
   □     □

17. Does any person proposed for insurance now have, or has he ever had:
   □     □
   a. Disease or disorder of lungs, asthma, emphysema, bronchitis or tuberculosis?
   □     □
   b. Disease or disorder of the heart or blood vessels, pain or discomfort in chest; high blood pressure, shortness of breath or heart murmur?
   □     □

18. In the past 5 years has any person proposed for insurance:
   □     □
   a. Had or been advised to have a surgical operation?
   □     □
   b. Had an electrocardiogram, X-ray, blood study, or other diagnostic study?
   □     □
   c. Been a patient in any hospital, institution, or sanitarium?
   □     □
19. Proposed Insured's Height and Weight (In shoes) _______________ inches _______________ lbs. past year

20. Has any person proposed for insurance been under Yes No treatment or taken medication in past 2 years?

21. Has any person proposed for insurance smoked one or more cigarettes within the last 12 months?

22. How much time has the proposed insured lost from work during the last two years because of illness or injuries? None Weeks

23. Has any person proposed for insurance ever:
   a. Applied for or received pension or disability benefits? No
   b. Been rejected for or received a medical discharge from Military Service? No
   c. Sought or received advice for, or treatment of, or been arrested for the use of alcohol, marijuana, or drugs? No
   d. Used amphetamines, barbiturates, hallucinogenics, narcotics, or marijuana unless administered on the advice of a physician? No
   e. Had any parents, brothers or sisters who had heart disease, diabetes, stroke or high blood pressure? No

24. Driving Record
Within the past 3 years has any person to be covered been convicted of or pleaded guilty to:
   a. Three or more moving traffic violations? No
   b. Driving under the influence of alcohol or drugs? No
   c. Or had a driver's license revoked or suspended? No

25. DETAILS OF "YES" answers. (List name and address of all doctors or hospitals plus date of attacks and recoveries.)

26. FAMILY RECORD - of Proposed Insured

<table>
<thead>
<tr>
<th>Name</th>
<th>Age if Living</th>
<th>Health - Reason If Not Good</th>
<th>Age at Death</th>
<th>Cause of Death</th>
</tr>
</thead>
<tbody>
<tr>
<td>Father</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mother</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brothers and Sisters</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I (we) represent that all statements and answers in this application, any supplements, additional forms or medical examinations required by the Company are to the best of our knowledge and belief true, complete and correctly recorded, and it is agreed (1) Such statements and answers are a part of the application and constitute the sole basis for issuance of any insurance hereunder; (2) No information known to any agent or employee of the Company shall be binding upon the Company unless written on and made a part of this Application; (3) EXCEPT AS STATED IN A DULY EXECUTED CONDITIONAL RECEIPT BEARING THE SAME PRINTED NUMBER AS THIS APPLICATION, A DULY EXECUTED GOVERNMENT ISSUED CONDITIONAL INSURANCE CERTIFICATE BEARING THE SAME PRINTED NUMBER AS THIS APPLICATION, OR A CONDITIONAL SALARY DEDUCTION AGREEMENT RELATING TO THE PROPOSED INSURANCE AND DELIVERED TO THE APPLICANT, THE COMPANY GRANTS NO INSURANCE UNDER THIS APPLICATION UNLESS AND UNTIL DURING THE CONTINUED INSURABILITY OF THE PROPOSED INSURED, AND ALL OTHERS PROPOSED FOR INSURANCE HEREBIN, A POLICY ISSUED ON THIS APPLICATION IS DELIVERED TO THE APPLICANT; AND THE FULL FIRST PREMIUM IS PAID; and (4) Acceptance of the issued policy will constitute ratification of changes made by the Company as Home Office Corrections or Additions except where written acceptance of non-administrative changes is required by law.

Signed at ____________________________ State of ____________________________ day of ____________________________ 19

No 28322

Signature of Proposed Insured (If over age 14)

Signature of Agent

Signature of Parent (Juvenile Policy Only)
Insurance Application

AGENT’S REPORT
(This information must be completed with EVERY application)

1. Was inspection or Telemark ordered? Date Attach Carbon Copies of Tickets

2. Residence Address of Proposed Insured: Street and No.

   HOME: Area Code  Number

   BUSINESS: Area Code  Number

3. Former Address (last 3 years)

4. Employer and Address

5. Maiden name of any woman to be insured if marriage occurred within five years.

6. How much insurance does spouse have? $  

EXAMINATION ARRANGEMENTS BY:

1. Is a medical examination being arranged for?  

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
</table>

2. On | Date |

3. Name of Doctor or Former:

4. Is the one of our approved examiners?

5. If rules require, did you arrange for:

   Second Medical exam?

   EKG?

   Chest X-ray?

   Sending specimen to Home Office Reference Lab?

   SMA-12 Blood Test?

IF PROPOSED INSURED IS UNDER AGE 15:

1. Did you see the child proposed for insurance?  

   |  Yes | No |

2. Does the child appear in good health?  

   |  Yes | No |

3. How long have you known the child?  

4. How many brothers and sisters has the child?  

5. Are all brothers and sisters insured?  

   |  Yes | No |

   If “No” give reasons.

   Amount of insurance in force on supporting parents? $  

   Remarks:

TO BE COMPLETED IF INSURANCE IS FOR BUSINESS PURPOSES OR FOR ESTATE CONSERVATION

1. Value of Business $ Proposed Insured’s Interest  %

2. Proposed Insured’s Compensation: Salary $ Bonus $ Other $  

3. Purpose of Insurance.
   - Key Person
   - Split Dollar
   - Fund Buy and Sell
   - Executive Bonus
   - Deferred Compensation
   - Estate Liquidation

4. Names of Other Stockholders, Officers and Partners and Amount of Business Insurance Carried on Their Lives.

<table>
<thead>
<tr>
<th>Name and Title</th>
<th>Value of Business interest</th>
<th>Percent owned</th>
<th>Amount now Carried</th>
<th>Amount now applied for</th>
</tr>
</thead>
</table>

If any not insured, give explanation.

1. Details to any of above questions (state question number).

   

SOURCE OF [Policyowner: 3Cold Canvas: 5Policyowner Service lead
PROSPECT: 2Referral lead: 4Direct Mail: 6Other

Credit this application to Other Agent

% Code No.  

THIS REPORT MUST BE SIGNED

Signature of Writing Agent

ANNUAL PREMIUM CALCULATION

<table>
<thead>
<tr>
<th>Premium</th>
<th>Units</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Ins</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>WP on Basic</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>Term Riders</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>WP on Riders</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>AGB</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>GROPPO</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>CIR</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>SIR</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>Supertal</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>WP% of Prem. (Un. Life)</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>Q.D.F. (Not Un. Life)</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>Total Annual Premium</td>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>Times ($X) Mode Factor (Not Un. Life)</td>
<td>$</td>
<td>X</td>
</tr>
</tbody>
</table>

I CONFIRM THAT I HAVE DELIVERED TO THE PROPOSED INSURED THE FAIR CREDIT REPORTING ACT AND MEDICAL INFORMATION BUREAU NOTICES, FORM 1453.
## Client Asset Sheet

**Name:**

**Date of Birth:**

**Name:**

**Date of Birth:**

**Children:**

**Date of Birth:**

**Date of Birth:**

**Date of Birth:**

**Date of Birth:**

### Assets

<table>
<thead>
<tr>
<th>Markets Value of Home</th>
<th>His</th>
<th>Hers</th>
<th>Joint</th>
<th>Indebt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other Real Estate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Personal Property</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checking Accounts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Savings Accounts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C.D.'s, Money Market</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stock, Securities</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Life Insurance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retirement Plan Death Benefit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Other Information

- **His Salary:**
- **Her Salary:**
- **Other Income:**
- **Rental Income:**
- **Consulting Income:**
- **Self Employ. Income:**

**Disability Insurance:**

**Health Insurance:**

**Income need in the event of a premature death:**

**Do you have a Will:**

**Inheritance potential:**

### Objectives
Estate Settlement Worksheet

1. Gross Estate
2. Less Debts, Outstanding Taxes, etc.
3. Less Probate and Administration Costs
4. Adjusted Gross Estate
5. Less Marital Deduction
6. Less Charitable Contributions
7. Taxable Estate
8. Tentative Federal Estate Tax
9. Less Federal Tax Credit
10. Approximate Federal Estate Tax
11. Total Cash Required

(a) The marital deduction does not apply if there is no surviving spouse.
(b) This amount can be either (1) the amount from line 4; (2) the amount necessary to bring estate tax to zero — Line 4 less $225,000 for 1982; $275,000 for 1983; $325,000 for 1984; $400,000 for 1985; $500,000 for 1986 and $600,000 for 1987 and later years or (3) any other amount.
(c) The estate must have sufficient funds to make the bequest.
(d) Use $82,200 for 1982; $78,000 for 1983; $76,300 for 1984; $121,800 for 1985; $155,500 for 1986 and $192,800 for 1987 and later years.
(e) Including maximum credit for state inheritance taxes. The state taxes may exceed this credit so the total federal and state taxes may exceed this estimate.
(f) Sum of lines 2, 3 and 10.

### Federal Estate Tax

<table>
<thead>
<tr>
<th>Taxable Estate</th>
<th>Estate Tax</th>
<th>% Tax in estate bracket</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10,000</td>
<td>$1,400</td>
<td>20%</td>
</tr>
<tr>
<td>$20,000</td>
<td>3,800</td>
<td></td>
</tr>
<tr>
<td>$40,000</td>
<td>8,200</td>
<td></td>
</tr>
<tr>
<td>$60,000</td>
<td>13,000</td>
<td></td>
</tr>
<tr>
<td>$80,000</td>
<td>18,000</td>
<td></td>
</tr>
<tr>
<td>$100,000</td>
<td>23,000</td>
<td></td>
</tr>
<tr>
<td>$150,000</td>
<td>35,000</td>
<td></td>
</tr>
<tr>
<td>$250,000</td>
<td>70,000</td>
<td></td>
</tr>
<tr>
<td>$500,000</td>
<td>151,000</td>
<td></td>
</tr>
<tr>
<td>$750,000</td>
<td>242,000</td>
<td></td>
</tr>
<tr>
<td>1,000,000</td>
<td>348,000</td>
<td></td>
</tr>
<tr>
<td>1,250,000</td>
<td>448,200</td>
<td></td>
</tr>
<tr>
<td>1,500,000</td>
<td>583,300</td>
<td></td>
</tr>
<tr>
<td>2,000,000</td>
<td>790,800</td>
<td></td>
</tr>
</tbody>
</table>

### Average Probate and Administration Costs

- Probate Estate
- Probate and Administration Costs

<table>
<thead>
<tr>
<th>Probate Estate</th>
<th>Probate and Admin. Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$25,000</td>
<td>$2,500</td>
</tr>
<tr>
<td>$50,000</td>
<td>$3,500</td>
</tr>
<tr>
<td>$100,000</td>
<td>$5,500</td>
</tr>
<tr>
<td>$200,000</td>
<td>$9,750</td>
</tr>
<tr>
<td>$300,000</td>
<td>$14,500</td>
</tr>
<tr>
<td>$400,000</td>
<td>$19,500</td>
</tr>
<tr>
<td>$500,000</td>
<td>$25,000</td>
</tr>
<tr>
<td>$600,000</td>
<td>$30,000</td>
</tr>
<tr>
<td>$700,000</td>
<td>$35,000</td>
</tr>
<tr>
<td>$800,000</td>
<td>$40,000</td>
</tr>
<tr>
<td>$900,000</td>
<td>$45,000</td>
</tr>
<tr>
<td>$1,000,000</td>
<td>$50,000</td>
</tr>
</tbody>
</table>

Upper limits of estate tax brackets will be reduced in years 1982-1985. Do not refer to tax brackets below the current year. The tax brackets in parentheses will become effective when the year indexed is reached. For example, in 1982 a taxable estate of $3,800,000 will be in a 61% marginal tax bracket. In 1983, the same size taxable estate will be in a 60% marginal bracket. In 1986, a taxable estate of this size or any taxable estate in excess of $2,500,000 will be in a maximum tax bracket of 50%.

Compare the reverse side to show Estate Settlemetn Data if the order of death is reversed.
# Cash Flow Statement

For the Year Ending ______________

<table>
<thead>
<tr>
<th>Cash Balance at Beginning of the Year</th>
<th>$ __________</th>
</tr>
</thead>
</table>

**INFLOWS**

- Salaries after taxes  
- Dividends in cash  
- Interest received

**TOTAL INFLOWS**  

**OUTFLOWS**

- Savings and Investment  
- Fixed Outflows
  - Mortgage note payments  
  - Automobile note payments  
  - Insurance premiums

**Total Fixed Outflows**  

- Variable Outflows
  - Food  
  - Transportation  
  - Utilities/household exp.  
  - Clothes and personal care  
  - Recreation and vacations  
  - Medical and dental care  
  - Taxes  
  - Miscellaneous

**Total Variable Outflows**

**TOTAL OUTFLOWS**  

| Cash Balance at End of the Year | $ __________ |
Pro Forma Cash Flow Statement

For the Quarter Ending _____________________

<table>
<thead>
<tr>
<th></th>
<th>$</th>
<th>$</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash balance at beginning of month</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>INFLows</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Salaries after taxes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dividends in cash</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interest income</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Borrowed Funds</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL INFLows</strong></td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td><strong>OUTFLows</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Savings and Investments</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed Outflows</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mortgage note payment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automobile note payment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insurance premiums</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable Outflows</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Food</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transportation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Household Expenses (inc. util.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clothes/personal care</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recreation and vacations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medical and dental care</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Credit card payments</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL OUTFLows</strong></td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td><strong>NET CASH POSITION</strong></td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>Additional funds needed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cash Balance at End of Month</td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
</tbody>
</table>
## Statement of Financial Position

**As of __________**

<table>
<thead>
<tr>
<th><strong>ASSETS</strong></th>
<th><strong>LIABILITIES AND NET WORTH</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cash and Cash Equivalents</strong></td>
<td><strong>Liabilities</strong></td>
</tr>
<tr>
<td>Cash and checking account</td>
<td>Credit card balance</td>
</tr>
<tr>
<td>All-savers Certificate</td>
<td>Automobile note bal.</td>
</tr>
<tr>
<td>Money Market Fund</td>
<td>Mortgage note bal.</td>
</tr>
<tr>
<td></td>
<td>Total Liabilities</td>
</tr>
</tbody>
</table>

**Total Cash/Cash Equiv.**

<table>
<thead>
<tr>
<th><strong>Invested Assets</strong></th>
<th><strong>Net Worth</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Stocks and bonds</td>
<td></td>
</tr>
<tr>
<td>Life insurance cash value</td>
<td></td>
</tr>
<tr>
<td>Vested portion of pension</td>
<td></td>
</tr>
<tr>
<td>Total Invested Assets</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Use Assets</strong></th>
<th><strong>Total Use Assets</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Residence</td>
<td></td>
</tr>
<tr>
<td>Automobiles</td>
<td></td>
</tr>
<tr>
<td>Household furnishings</td>
<td></td>
</tr>
<tr>
<td>Clothing, jewelry, etc.</td>
<td></td>
</tr>
<tr>
<td>Total Use Assets</td>
<td></td>
</tr>
</tbody>
</table>

**TOTAL ASSETS**

<table>
<thead>
<tr>
<th><strong>TOTAL LIABILITIES AND NET WORTH</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TOTAL LIABILITIES</strong></td>
</tr>
<tr>
<td>-------------------------------------</td>
</tr>
</tbody>
</table>

157
References


The Design of a Small Business Database using the Semantic Database Model.

by

Jac F. Morgan

B.S., Kansas State University, 1975
M.S., Kansas State University, 1978

AN ABSTRACT OF A MASTER'S REPORT

Submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985
The implementation of a database management system requires a comprehensive understanding of the data used by an enterprise and of how the data is used. The first step in database design is to gather information about the data. The second step is to arrange this information into some kind of meaningful representation of the enterprise which will guide the logical and physical design of the database.

This report describes the Semantic Database Model (SDM), a tool for creating a conceptual model (meaningful representation) of the application environment. The SDM will model the information requirements of a small business, an insurance agency. The authors of the SDM have provided a database definition language (DbDL) which is to be used to describe the application environment. This report is written with the specific intent of providing a tool to teach undergraduate and graduate students how to use the SDM. Thus the DbDL is presented in the form of a user's manual. Suggested steps in the application of the SDM are provided and the use of these steps is demonstrated. The result of the application of these steps is an SDM schema which describes most of the insurance agency. A brief section will describe a method for converting the SDM model into a relation model of the enterprise. A discussion of the problems encountered when applying the SDM is provided.