A Personal Data Manager[1]

Peter Lyngbaek
Dennis McLeod

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0782

Abstract

The Personal Data Manager (PDM) is a simple database system for personal computers. PDM is intended to provide personal information management capabilities for the large class of personal computer users who are not computer experts, and who have no programming experience. PDM simply attempts to make a personal computer serve as an extension of its user's memory. PDM is based on a simple conceptual database model that includes high-level semantic modeling constructs, such as *objects*, *object kinds* (types), *attributes*, and *object frames*. A prescriptive user interface allows the contents of the database and the structure on the information in the database to be changed dynamically. A *working kind* is a run-time collection of database objects defined via the user-interface; working kinds can be interactively restricted and expanded and made part of the permanent database. This paper discusses the design of the Personal Data Manager, including the conceptual information model, the user interface, and a prototype implementation.

[1]This research was supported, in part, by the Danish Natural Science Research Council under grant 11-4132, a grant from the IBM Corporation, and by the Joint Services Electronics Program through the Air Force Office of Scientific Research under contract F49620-81-C-0070.

Proceedings of the Tenth International Conference on Very Large Data Bases.

1. Introduction

The past several years have seen a dramatic proliferation of personal computers, and anticipated future technological advances will continue to increase their power and reduce their cost. These flexible tools are improving the accessibility of computing resources for end-users who are not computer experts, and who may have little or no programming experience. While personal computers support a wide variety of specific applications, perhaps one of the most exciting and far-reaching potential uses of a personal computer is a general-purpose information manager and a tool for information sharing/communication.

The class of potential end-users of a personal information manager is mostly dominated by the large number of "home computer" users who are by no means computer experts, and who have little or no programming experience. Typical end-users may use an information system to manage personal data from a wide variety of applications, such as a phone directory, a calendar, an entertainment guide, a recipe file, a wine list, an index of vacation slides, etc. Novice end-users, however, may successfully utilize an information system only if it provides a simple and easily understandable interface that supports database communication and interaction in a straightforward manner. Potential end-users of a personal data manager also include professionals and engineers who have needs for capabilities to manage office information and design data.

In contradistinction to the principles and mechanisms of current production and research prototype database systems, a personal information management environment is best described by the following characteristics:

- The structural framework for information classification and logical access must be highly dynamic as the structure of the

Singapore, August, 1984

stored information may change over the lifetime of the database.

- The amount of structural information (e.g., kinds of data, kinds of inter-relationships, etc.) is large relative to the size of the database.

- The end-user is familiar with the application environment, but is not assumed to have expertise in databases or programming. In consequence, the end-user must serve as designer, accessor, and evolver of information, and is thus "the only" expert.

- Data of different modalities and varying length must be accommodated.

- The amount of information is relatively small.

This paper describes research on the *Personal Data Manager* (PDM), a simple database system for personal computers that allows novice end-users to directly define, classify, interrelate, and manipulate a universe of information objects. PDM is intended to make a personal computer serve as an extension of its user's memory. The critical research subtopics here are: a simple, but semantically expressive conceptual information model; an approach to information communication with the user (user interface); and a strategy for efficient information storage and access, which does not require external database design expertise.

The PDM conceptual information model includes a simple set of modeling constructs: *objects*, *types*, *attributes*, and *object frames*. A prescriptive user interface, closely coupled to the conceptual information model through the notion of a *working kind*, allows novice end-users to dynamically change the contents and structure of their databases. An experimental prototype implementation of the Personal Data Manager is currently underway.

The remainder of this paper is structured as follows. Section 2 contains a brief overview of related research. The Personal Data Manager conceptual model is defined in Section 3. Section 4 contains an example database application, to motivate the description of the PDM operations and user interface given in Section 5. The prototype implementation is described in Section 6. Finally, Section 7 describes conclusions and future research directions.

2. Background

During the last several years, the popularity of personal computing has experienced a tremendous growth. Almost daily new personal computers and software products for personal use are introduced. Notably, database and information management systems for micro computers are among the most popular products in any computer store. More than forty database products alone [Wells 84] run on the IBM Personal Computer (IBM PC) and compatible computer systems.

Current database systems available for personal data management on inexpensive micro-computers organize data as files of fixed-format records. For example, systems such as dBASE-II [Ashton-Tate 81] are based on record-oriented database models. Such record-oriented database systems have limited flexibility [Kent 79] in that once the record format is set up by a (possibly unexperienced) user, it is very difficult to alter the structure of the database. In addition, these data managers introduce a number of troublesome limitations, such as the maximum number of data fields per record and their fixed sizes.

The Personal Data Manager is based on a semantic database model. Semantic database models [Abrial 74, Chen 76, Smith 77, Hammer 81, Shipman 81, King 82a] and similar models for knowledge representation in Artificial Intelligence [Roussopoulos 75, Mylopoulos 80] introduce a semantically rich set of structuring primitives that support abstractions such as object types (for object classification), supertypes/subtypes (for generalization and specialization), and attributes (for inter-object mappings). Even though semantic database models in general are simpler and easier to understand than conventional database models (the network, hierarchical, and relational model), most current research approaches are relatively complicated.

The PDM user interface draws most directly on recent work in browsing-oriented database user interfaces [Cattell 80, Herot 80, Stonebraker 82, Wong 82]. The interface must be simple so that it can be understood by a novice user and allow that user to utilize the manipulation primitives of the data manager. Moreover, incremental learning of more complex features of the model must be supported. The approach taken here is based in part on experience gained with a prescriptive conceptual design and evolution methodology devised for databases defined with the event database model [King 82b]. Further experience was gained in an experimental user interface for the database model SDM [Hammer 81], called a *transaction specification*

Proceedings of the Tenth International Conference on Very Large Data Bases.

Singapore, August, 1984

advisor (TSA). The purpose of TSA is to guide an end-user in understanding the content and structure of a database, and formulating a transaction on that database [McLeod 82]. As such, the TSA provides prescriptive guidance to the user in browsing a database and querying it.

Although semantic database models have been considered the state-of-the-art in data model research for a number of years, only a few complete implementations have appeared. The most significant are the ADAPLEX implementation effort [Chan 82] underway at the Computer Corporation of America, and the Relationship-Entity-Datum Data Model [Cattell 83] implemented at the Xerox Palo Alto Research Center (PARC). The PDM prototype is an attempt to implement a semantic data model in a micro-computer environment characterized by limited resources (main memory and disk space).

3. The PDM Conceptual Model

The user views a PDM database as a collection of *objects*. Objects correspond to concepts, entities, or things with an associated meaning about which information is to be recorded. Examples include persons, friends, addresses, checks, recipes, and appointments. Each object is represented in the database as an atomic string of characters. The character strings are either user-defined or system-defined; they are displayable and serve as external object references. When referring to an object by its character string representation, a quoted string is used throughout this paper, e.g., "$455.75", "Mann's Chinese Theatre", and "(213) 743-2745".

An *object kind* (object type) is a named collection of database objects that share common properties. The name of a kind must be unique with respect to other kind names. Kinds are organized as a forest of trees, consisting of *basic kinds* and *specialized kinds*. A basic kind is a root node containing all the objects in the tree. A specialized kind is a subkind of a basic kind or another specialized kind; it contains a subset of the objects in its superkind. Object kinds are defined during database setup, and the user can dynamically define new basic kinds and specialized kinds. A specialized kind is specified by identifying existing kinds, and refining them in a stepwise manner to define a new kind, thereby identifying a collection of objects presumably sharing some common properties. This "snapshot" of objects is bound at the time it is created by the user.

Objects must be unique within a kind tree. This means that there can only be one object with a given character string representation in each basic kind. However, it is not necessary to require all objects in a database to have unique character string representations. Objects in different kind trees can correspond to different concepts or entities; however, this does not prevent them from being represented identically with respect to names. For example, the basic kinds Persons and Movies may both contain an object "Gandhi"; here, it is clear from the context if "Gandhi" is a person or a movie.

New objects can be created at any level in the kind tree. When an object is created as being of a certain kind, the object assumes all superkinds of that kind. However, the user must explicitly specify which other subkinds include the object. The object cannot automatically assume the correct set of subkinds, since subkinds are defined as snapshots rather than by predicates that are evaluated for the new object.

Each kind has an associated collection of *attributes*, which indicate the common aspects of the objects of that kind. An attribute has a name that is unique with respect to other attributes of the same kind. At any given time, each object of a kind has a (possibly null) value for its attributes, taken from the *value kind*; the *value* of an attribute can be single-valued or multi-valued. In effect, an attribute is a mapping from an object kind to its value kind. Attributes are inherited down the kind tree, so that attributes defined on a basic kind are defined on all the kinds in the tree. Each attribute has an inverse attribute called the *inverse*. The inverse of an attribute A from object kind K1 to value kind K2 is an attribute denoted by inv(A) from object kind K2 to value kind K1. An inverse attribute may optionally be given a user-defined name.

An *object frame* is a specification of the scope of the objects of a certain kind. It is used as an interface for input and output of objects of that kind, and can also be used as a unit of object protection, sharing, saving/restoring, etc. When used for object creation and update (input), the object frame specifies which object attribute values must be supplied. When used for object display (output), an object frame specifies which attribute values to display, and the format of the display. Not only attributes of the kind for which the frame is being defined may be specified, but also attributes of attribute values of that kind can be specified. In other words, it is possible to specify attributes at an arbitrary level of nesting. Object frames are uniquely named and any number of object

frames may be defined for an object kind; different frames may be provided for different purposes. Each object kind has a default object frame that specifies every attribute of the kind.

At the PDM user interface, a distinction is made between *permanent kinds*, a single *working kind*, and *temporary kinds*. Permanent kinds are stored permanently in long-term memory; they survive from one user session to another. The working kind is similar to a program variable that contains a set of objects. The contents of the working kind can be examined and changed via the user interface as described below. The working kind is always associated with a single permanent kind, called the *associated kind*, which is its immediate superkind in the kind tree. The working kind is said to be bound to its associated kind. The working kind may be made a permanent subkind of its associated superkind. This feature allows a user to dynamically alter the structure of a database. A temporary kind is a named instance of the working kind. At the time of creation, the temporary kind is bound to the associated kind of the working kind. Temporary kinds and the working kind inherit all the attributes of their associated kinds.

PDM provides a number of "built-in" predefined basic kinds called *simple kinds*. This includes *names*, which are string of varying length, *places*, *dates*, *time*, *telephone numbers*, *money*, etc. Simple kinds are treated as any other basic kinds, in that objects may be added to and removed from a simple kind, and specialized kinds may be defined. Simple kinds do not have any predefined attributes.

4. An Example Personal Database

To illustrate the operation of the Personal Data Manager, an example is provided here to show how the capabilities of the system can be used to define, use, and modify a database. The following section contains a detailed description of the individual PDM operations.

Consider a personal database the purpose of which is to keep track of data on persons and entertainment. To establish such a database, the user might first create a basic kind of objects called PERSONS. Then, to specify what kinds of information are to be recorded for each PERSON object, three attributes might be defined, which specify the Name, Address, and City in which the PERSON resides. When defining these three attributes, the user would specify the value kinds of the attributes, which would be PERSON NAMES, ADDRESSES, and CITIES, respectively; in this way, the specific nature of the relationship between a PERSON and his or her name and address would be specified. Next, several PERSON objects might be created, specifying values for their attributes.

In a similar way, the user might create an ENTERTAINMENT kind, which is intended to include information about films and restaurants. The data that might be recorded about each variety of entertainment could include the Name of the establishment and its Telephone Number. Further, the user might then specify two specialized kinds of entertainment, by defining two specialized kinds, namely RESTAURANTS and THEATRES. These two kinds of entertainment are distinguished because each has distinct associated attributes. RESTAURANTS may have an associated list of Specialties (with values from FOOD SPECIALTIES), while THEATRES might have a Current Film Title, a Price, and a list of Showtimes. Objects corresponding to restaurants and theatres can now be created.

Now suppose that it is desired to rate the restaurants according to the opinions of PERSONS, say using a one-to-five star rating scale. To do this, the user simply creates a new kind that could be called RESTAURANT RATINGS, which has the attributes Rater (a PERSON), Restaurant Rated (a RESTAURANT), and a Rating (whose value is from FIVE STAR SCALE). Ratings of restaurants by individual persons can then be entered.

Suppose now the user wants to distinguish among the kinds of PERSONS he/she knows, namely FRIENDS and BUSINESS ASSOCIATES. To do this, the objects of the kind PERSONS are first assigned to the working kind and then refined to include only those PERSONS who are FRIENDS. The working kind can now be made permanent, with the name FRIENDS. The user might next create a new attribute of FRIENDS, Home Telephone, which is specific to FRIENDS as opposed to PERSONS in general. Similarly, BUSINESS ASSOCIATES could be created as a specialized kind of PERSONS, with the attribute Business Phone. Note that the same PERSON object can be both a FRIEND and a BUSINESS ASSOCIATE.

Figure 4-1 shows the object kinds and attributes that have been defined in the example up to this point. Specifically, the following basic kinds have been defined: PERSONS, ENTERTAINMENT, RESTAURANT RATINGS, FOOD SPECIALTIES, FIVE STAR SCALE. The kind FIVE STAR SCALE will contain the following five objects: "*", "**", "***", "****", and "*****". The following specialized kinds have been defined: RESTAURANTS,

kind PERSONS
attributes
  Name: PERSON NAMES,
  Address: ADDRESSES,
  City: CITIES;

kind FRIENDS
superkind PERSONS
attributes Home Phone: TELEPHONE NUMBERS;

kind BUSINESS ASSOCIATES
superkind PERSONS
attributes Business Phone: TELEPHONE NUMBERS;

kind ENTERTAINMENT
attributes
  Name: NAMES,
  Telephone Number: TELEPHONE NUMBERS;

kind RESTAURANTS
superkind ENTERTAINMENT
attributes Specialties: FOOD SPECIALTIES;

kind FOOD SPECIALTIES

kind THEATRES
superkind ENTERTAINMENT
attributes
  Current Film Title: NAMES,
  Price: MONEY,
  Showtimes: TIMES;

kind RESTAURANT RATINGS
attributes
  Rater: PERSONS,
  Restaurant Rated: RESTAURANTS,
  Rating: FIVE STAR SCALE;

kind FIVE STAR SCALE

Figure 4-1: The Structure of the Example Database

THEATRES, FRIENDS, and BUSINESS ASSOCIATES. The pre-defined simple kinds ADDRESSES, CITIES, NAMES, MONEY, PERSON NAMES, TELEPHONE NUMBERS, and TIMES have been used as value kinds.

The user is now ready to request information from the database. Suppose the user wants to know the names of all the Sushi restaurants to which a friend Yoko Hibi gave a four or five star rating. A natural way to find the desired information would be to assign the objects of FRIENDS to the working kind, and then restrict the working kind to contain only those friends with the name "Yoko Hibi". In order to find the restaurant ratings for which Yoko Hibi is the rater, the inverse of the attribute Rater is applied to the object(s) in the working kind. Then all Yoko Hibi's restaurant ratings are restricted to include only those with rating "*****" or "******". Now the attribute Restaurant Rated is applied to the objects in the working kind. This returns all the restaurants that the friend Yoko Hibi rated with four or five stars. To finish the query, the restaurants must be restricted to include only those with specialty "sushi"; and the attribute Name of each qualifying object is then applied. Now the working kind contains the names of the Sushi restaurants that the friend Yoko Hibi gave a four or five star rating. In order to see the result of the query the objects of the working kind must be printed.

Note that the working kind in the above example moves from FRIENDS to RESTAURANT RATINGS to RESTAURANTS to NAMES, and how it is restricted in each step by a predicate on one of the attributes. However, there are of course several alternative query formulations, e.g., the working kind might be initialized to RESTAURANTS first, and then restricted to Sushi restaurants, etc.

5. The PDM User Interface

The capabilities provided for the end-users to manipulate a PDM database can be categorized as follows:

1. Operations for working kind manipulation are used for retrieving objects from the database into the working kind.

2. Operations for object manipulation are used for creating new objects and manipulating the objects in the working kind.

The PDM operations support a user-friendly interface that employs prescriptive guidance and layered functionality to assist novice users in utilizing features of the system with which they are not familiar. For example, if a non-existing entity (object, kind, attribute, or frame) is referenced during database creation or modification, the user has the option of creating a new entity or specifying an existing entity.

For the purpose of this paper a straightforward line-oriented user interface is utilized to demonstrate the functionality of the PDM operations. The actual interface prototype will however make extensive use of graphics capabilities and pointing devices. The PDM operations are detailed below.

5.1. Operations for Working Kind Manipulation

The operations for working kind manipulation allow a user to interactively specify the objects contained in the working kind. Those objects are the target for the object manipulation operations described in the next section. As mentioned before, the working kind is always bound to a permanent kind (the associated kind). The working kind contains a subset of the objects of its associated kind, and the attributes of the associated kind are inherited by the working kind. The user interface includes the following operations for working kind manipulation:

Initialize — The working kind may be initialized by specifying an associated object kind and an initial set of objects. The working kind can be initialized to the empty set, to a set of explicitly enumerated objects, to a set of objects that satisfy a certain constraint, or to the set of all objects of a specified kind.

Expand — The working kind may be expanded with a set of objects (the expanding set). The union of the working kind and the expanding set will be the new value of the working kind. The expanding set is a subset of a permanent kind, and is either a set of explicitly enumerated objects, a set of objects that satisfy a certain constraint, or the set of all objects of a specified kind. The expansion causes the working kind to be bound to the closest common superkind of the associated kind and the permanent kind of the expanding set.

Restrict — The working kind may be restricted by a set of objects (the restricting set). The intersection of the working kind and the restricting set will be the new value of the working kind. The restricting set is a subset of a permanent kind, and is either a set of explicitly enumerated objects, a set of objects that satisfy a certain constraint, or the set of all objects of a specified kind. When the working kind is restricted, its associated kind remains the same.

Remove — A set of objects may be removed from the working kind. The set difference between the working kind and the set of removed objects will be the new value of the working kind. The set of removed objects is a subset of a permanent kind, and is either a set of explicitly enumerated objects, a set of objects that satisfy a certain constraint, or a set of all objects of a specified kind. When objects are removed from the working kind, its associated kind remains the same.

Map — The working kind may be mapped to the value set of one of its attributes. The set of objects in the value kind of the mapping attribute that are values of objects in the working kind will be the new value of the working kind. The mapping operation causes the working kind to be bound to the value kind of the mapping attribute.

Name — The working kind may be named, which causes a temporary kind to be created as a subkind of the associated kind. The name of the temporary kind must be unique with respect to the names of other kinds, permanent as well as temporary. The temporary kind is a snapshot of the working kind at the time it is created. The naming operation does not change the content of the working kind.

Display Context — The context of the working kind may be displayed. In this way, it is possible to examine the position of the working kind in the kind hierarchy and to determine which attributes and object frames are defined.

Figure 5-1 illustrates the use of the working kind manipulation operations. First, the working kind is initialized to contain the three objects "Georg", "Hamideh", and "Victor", and the set is made a temporary kind by the name FRIENDS. Then the working kind is expanded to contain all the objects of PERSONS except "Linda" and "John". Now the working kind is restricted to contain only those objects representing persons living in Los Angeles and that set is made a temporary kind by the name BUSINESS

ASSOCIATES. The intersection of FRIENDS and BUSINESS ASSOCIATES is then created. Finally, the context is displayed.

The operations described above, allow a user to select any subset of the objects in a PDM database. Temporary kinds are useful for database browsing. Suppose that the content of the working kind, at a given point in time, reflects a situation a user might want to reestablish at some future time. By naming the working kind, the user can later initialize the working kind to contain the objects of the temporary kind and thus restore the previous content.

```
>initialize
    associated kind: PERSONS
    type of initialization: enumerate
    members: Georg, Hamideh, Victor
>name
    temporary kind: FRIENDS
>expand
    type of expansion: kind
    expanding kind: PERSONS
>remove
    type of removal: enumerate
    objects to be removed: Linda, John
>restrict
    type of restriction: predicate
    restricting predicate: City = Los Angeles
>name
    temporary kind: BUSINESS ASSOCIATES
>initialize
    associated kind: PERSONS
    type of initialization: kind
    initializing kind: FRIENDS
>restrict
    type of restriction: kind
    restricting kind: BUSINESS ASSOCIATES
>name
    temporary kind: FRIENDS AND ASSOCIATES
>display context
    the associated kind is PERSONS
    the temporary subkinds are
        FRIENDS
        BUSINESS ASSOCIATES
        FRIENDS AND ASSOCIATES
    the attributes are
        Name: NAMES
        Address: ADDRESSES
        City: CITIES
    the object frame is Name
>
```

Figure 5-1: Examples of Working Kind Manipulation

5.2. Operations for Object Manipulation

The PDM operations for object manipulation allow a user to modify the content of a database. The operations, in general, operates on the entire set of objects of the working kind. The user interface includes the following object manipulation operations:

Create	A new object may be created as a permanent member of the associated kind. The character string representation of the object and its attribute values are specified. If an object frame is specified, attribute values must be supplied as specified by the frame. The new object will also be included in the working kind, and it may be specified to be a member of other kinds in the kind hierarchy.
Restore	New objects may be loaded into a PDM database from an external file. The corresponding object kind, an object frame, and the file containing the object values must be specified.
Delete	The objects contained in the working kind may be deleted from the associated kind and all its subkinds. The superkind of the associated kind, however, is not affected. If the associated kind is a basic kind, the objects are deleted entirely from the database. The attribute values of the objects being deleted are also deleted if they are of a simple kind and not in the value set of any other attributes.
Modify	The character string representation of the objects contained in the working kind may be modified. If an object frame is specified, the values of the attributes in the frame may also be modified.
Display	The objects in the working kind together with their attribute values may be displayed. If no object frame is explicitly specified, the values of all the attributes are output, otherwise attribute values are output as specified by the object frame.

Print	The objects in the working kind together with their attribute values may be printed. Attribute values are printed as described for the Display operation.
Save	The objects in the working kind together with their attribute values may be output to an external file. Attribute values are output as described for the Display operation.
Add	The objects in the working kind may either be added to a an existing subkind of the associated kind or assigned as the initial value of a new permanent subkind created by the Add operation. If an existing subkind is specified, the union of that subkind and the working kind will be the new value of the specified subkind.

The operations illustrated in Figure 5-2 are based on the full schema of the Example Personal Database shown in Figure 4-1. First, the object "Pelican's Catch" is created as a member of RESTAURANTS. The food specialty "oyster" is created on-the-fly as a member of FOOD SPECIALTIES. The restaurant Yagura Ichiban is then created as the object "Yoko's favourite". Next, all the restaurants ratings of Norm Freeman are deleted from the database. Finally, the PERSONS "John" and "Linda" are added to the specialized kind BUSINESS ASSOCIATES and then made a permanent subkind by the name GOLF PARTNERS.

The operations described above provides a browsing oriented user interface tightly coupled to the PDM information model through the notion of a working kind and temporary kinds. It is easy to restructure a PDM database. The operations for working kind manipulation together with the Add and Delete operations allow a user to move objects up and down the kind hierarchies. The Save and Restore operations provide a way to communicate objects from one PDM database to another and allow external programs to produce and consume PDM objects.

6. Meta-Data as PDM Objects

A very important aspect of a personal information management system is the support provided for defining and modifying the structure of the data in the database. Novice end-users cannot be expected to define the "right" database structure at their first attempts. Therefore, it should be as simple to modify meta-data (that is, the data describing the structure of the database) as it is to modify the data in the database.

```
>initialize
    associated kind: RESTAURANTS
    type of initialization:
>create
    object frame: default
    member of RESTAURANTS: Pelican's Catch
        Name: Pelican's Catch
        Telephone Number: 450-2983
        Specialties: oyster
        is oyster a new object? yes
>create
    object frame: default
    member of RESTAURANTS: Yoko's favourite
        Name: Yagura Ichiban
        Telephone Number: 623-4141
        Specialties: sushi, tempura
>initialize
    associated kind: RESTAURANT RATINGS
    type of initialization: kind
    kind name: RESTAURANT RATINGS
>restrict
    type of restriction: predicate
    restricting predicate: Rater.Name = Al Jones
>delete
>initialize
    associated kind: PERSONS
    type of initialization: enumerate
    members: John, Linda
>add
    subkind of PERSONS: BUSINESS ASSOCIATES
>add
    subkind of PERSONS: GOLF PARTNERS
    is GOLF PARTNERS a new permanent kind: yes
>
```

Figure 5-2: Examples of Object Manipulation

PDM supports meta-data manipulation simply by treating meta-data as PDM objects. The meta-data is described in the database by a number of kinds, so called meta kinds, which are manipulated as ordinary object kinds. That way, all the operations introduced in the previous section may be applied to meta-data. A PDM database contains the following meta kinds:

- KINDS contains all the object kinds defined in the database. KINDS has a number of subkinds describing basic kinds, specialized kinds, and simple kinds.

- ATTRIBUTES contains all the attributes defined on the object kinds in the database.

- FRAMES contains all the object frames defined for the objects in the database.

Figure 6-1 shows how the meta kind ATTRIBUTES is defined. ATTRIBUTES has a number of meta attributes. These meta attributes describes the properties of the user-defined attributes. A user-defined attribute is described by the object kind on which the attribute is defined, its value kind, its attribute type (single-valued, multi-valued, etc.), and its inverse attribute.

meta kind ATTRIBUTES
meta attributes
  Domain: KINDS,
  Value kind: KINDS,
  Type: ATTRIBUTE TYPE,
  Inverse: ATTRIBUTES;

Figure 6-1: The Meta Kind ATTRIBUTES

The meta kinds are created and manipulated as ordinary kinds. For instance, a new attribute may be added to an object kind K simply by adding a value to the meta attribute Attributes of K. The new attribute may be added by using the modify operation. This suggests that the PDM operations are generic. For example, the create operation may be used to create a person, a kind of persons, a property of persons, etc. Figure 6-2 shows how the object kind RESTAURANTS is created. First, the working kind is bound to the meta kind KINDS. That way, the create operation will create a new object kind. The new member of KINDS is called RESTAURANTS. Its superkind is ENTERTAINMENT, and it has a multi-valued attribute Specialties with value kind FOOD SPECIALTIES and the inverse Specialty of. No object frame is defined for the kind RESTAURANTS.

```
>initialize
   associated kind: KINDS
   type of initialization:
>create
   new member: RESTAURANTS
   Superkind: ENTERTAINMENT
   Attributes: Specialties
      Value kind: FOOD SPECIALTIES
      Type: multi-valued
      Inverse: Specialty of
   Attributes:
   Frames:
>
```

Figure 6-2: Creation of a New Object Kind

7. The PDM Prototype Implementation

A prototype implementation of the Personal Data Manager has been designed for the IBM PC, and is currently under development. The PDM is being implemented in IBM Personal Computer Pascal [IBM 81] running in the environment of the IBM Personal Computer Disk Operating System (DOS) [IBM 82]. The prototype requires a minimum of 128K bytes of main memory. Like other micro computers, the IBM PC is limited by its small main memory and relatively slow access to data stored on its floppy disks. In consequence, a major aspect of the design philosophy is to compact main memory and speed up access to the disk.

Each object in a PDM database is internally distinguished by a unique *object key* that serves as a handle on the object. Natural numbers have been chosen as object keys; they are simple to generate and allow object kinds (sets of objects) to be represented as bit vectors, a representation that under certain circumstances is very efficient in terms of storage space. Object kinds and attributes are also internally distinguished by unique keys (*kind keys* and *attribute keys*) that are natural numbers.

The PDM prototype implementation maintains four categories of data structures:

1. representation of individual objects,

2. representation of attribute values,

3. representation of kinds (collections) of objects,

4. representation of kind hierarchies and attributes (meta-data).

All objects in a PDM database are stored in an Object B-tree indexed on the objects character string representation. The leaf nodes associate the character strings with the keys of the objects they represent. Notice that objects in different basic kinds may have the same character string representation. For example, the object "Gandhi" of the kind Persons may have the object key 743, and the object "Gandhi" of the kind Movies may have the object key 1244. In that case, the Object B-tree associates the character string "Gandhi"

with the set {743, 1244}. The Object B-tree is stored on the disk and the first levels of the tree are also maintained in the main memory, thereby limiting the number of disk accesses required to find a given object. Note that every database object is stored symbolically in the Object B-tree. At the physical level, no distinction is made between objects of different kinds such as integers, reals, and booleans that typically are stored in an encoded representation. Thus, the number "1984" is stored as the four characters "1","9","8","4" and not as an integer in a single memory word.

The prototype must also support the mapping from object keys to their corresponding character string representations. A table indexed on object keys contains pointers to the leaf nodes of the object B-tree. For example, if the object "(213) 452-1031" has the object key 2486, then the array element with index 2486 contains a pointer to the string "(213) 452-1031" in the object B-tree. The table is maintained in main memory. Thus, given an object key, a reference to the corresponding object can be obtained by just a single table lookup, and the object itself can be obtained by just a single disk access.

The attribute values of individual objects are also stored in a B-tree. The Attribute B-tree is indexed on a combined object key and attribute key. The data are all the object keys of the objects in the corresponding attribute value. Suppose, for example, that an object with the object key 345 has a multi-valued attribute with the attribute key 465 and that the values of the attribute includes three objects with the object keys 9823, 5409, and 3339. In that case, the Attribute B-tree would map the combined key 345:465 to the set {9823, 5409, 3339}. Notice that inverses are explicitly stored in the Attribute B-tree. In the above example, the combined key 9823:465 would be mapped to {345}, and so would the combined keys 5409:465 and 3339:465. The Attribute B-tree is stored on the disk. Like the Object B-tree the first levels of the Attribute B-tree are maintained in main memory in order to reduce access time.

An object bit map stored on disk and (partially) maintained in main memory records which objects belong to which kinds. Each kind defined in the database has an associated bit vector that records all the objects of the kind. Bit x of bit vector y is on (1) if the object with object key x is a member of the kind with kind key, otherwise bit x is off (0). One bit vector at a time can be loaded into main memory As new objects are created, the bit vectors are extended. Notice how simple the working kind or a temporary kind is represented by a bit vector.

Typically, a kind only contains a small percentage of the total number of objects in a database. The corresponding bit vector therefore has a high percentage of off bits. A technique for compressing the bit vectors utilizes this fact to save storage space for the bit vectors. Each bit vector has a corresponding index vector. Bit i of the index vector is on if any of the bits (i-1)*C+1 to i*C in the bit vector is on. Thus, a single bit of the index map indicates if a group of C bits of the bit vector are all off. The compression is achieved by not storing the groups of C bits that are all off. The optimal value of the constant C depends on the relative frequency of off bits in the bit vector.

The last set of data structures is used to represent meta-data. Object kinds are represented as a tree. Each node contains the name of a kind, the kind key (pointer to the bit vector), pointers to the superkind and the subkinds, and a list of attribute keys. The attributes are represented in a table. Each entry describes an attribute by its name, attribute key, and the key of its value kind. Both the kind tree and the attribute table are maintained in main memory during run-time.

8. Conclusions and Future Directions

This paper has described the Personal Data Manager, a simple database system for novice end-users. First, the PDM conceptual information model was described. Then, an example database application was used to motivate a description of the PDM operations. Finally, a PDM prototype implementation was discussed.

The Personal Data Manager (PDM) is an approach to storing and retrieving data on a personal computer. PDM is not a record manager. Rather, it provides a simple view of data, in terms of information objects, kinds of objects, and attributes of objects. Objects can be directly related to one another; this eliminates the need for storing multiple copies of data and allocating storage for data that is not yet in the database. Contrary to current record-oriented database systems for personal computers, the Personal Data Manager does not limit the size or number of objects, as long as the host computer has memory capacity to store them. The PDM conceptual information model is a binary data model [Abrial 74, Bracchi 76]. Therefore, n-ary relationships (for arbitrary values of n greater than 2) cannot be modeled directly. However, any n-ary relationship can easily be modeled by introducing an artificial relationship object as described in [Lyngbaek 84].

The Personal Data Manager is intended to be easy to use; integrated help facilities and guidance to teach the user are provided by a user-interface that is closely coupled to the conceptual information model through the notion of a working kind. The structure of the information in a PDM database can be changed dynamically as new kinds and patterns of data become important; in PDM it is just as easy to add a new attribute as it as to add a new attribute value. The Personal Data Manager has been designed primarily for small personal computers. However, the ideas and techniques of PDM are also applicable to larger computer systems.

Analysis, testing, and extensions of the research described in this paper are currently under study. In particular, the PDM prototype, that at the time of this writing is being implemented, will be used to further assess the adequacy and completeness of the model and its operations. In its present form, the Personal Data Manager does not support format control. Furthermore, at the user interface, a distinction is made between meta-data and ordinary data. This distinction can be avoided by introducing the meta-kinds KINDS, ATTRIBUTES, and FRAMES. By treating meta-data as PDM objects, a user may query and create meta-data and ordinary data in a uniform manner. For example, a new object kind is created as a member of KINDS with (meta) attributes Superkind, Attributes, and Frames. Both format control and a completely uniform representation of meta-data and ordinary data integrated with the concept of the working kind have been planned. It is also intended that the PDM operations will be embedded into a programming language. This will allow end-users to write application programs that may access a PDM database. In order to manage such application programs, it should be possible to store them as information objects in the database.

Finally, a distributed version of the Personal Data Manager has been planned. A network of PDMs will be based on the ideas and results described in [Lyngbaek 83]. The goal is a personal information management environment that allows users of highly autonomous databases to communicate with each other, access remote information objects, and in a controlled manner share information objects. It is also intended that the distributed PDM will serve as an access port to large database systems by providing them with PDM interfaces.

References

[Abrial 74] Abrial, J. R.
Data Semantics.
In Klimbie, J. W. and Koffman, K. L. (editors), *Data Base Management*, , pages 1-59. North-Holland, Amsterdam, 1974.

[Ashton-Tate 81] *dBASE II Assembly Language Relational Database Management System User Manual*
Ashton-Tate, Los Angeles, Ca., 1981.

[Bracchi 76] Bracchi, G., Paolini, P., and Pelagatti, G.
Binary Logical Associations in Data Modelling.
In *Proc. IFIP TC-2 Working Conference on Modelling in Data Base Management Systems*, pages 125-148. 1976.

[Cattell 80] Cattell, R. G. G.
An Entity-Based Database User Interface.
In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. Santa Monica, Ca., May, 1980.

[Cattell 83] Cattell, R. G. G.
Design and Implementation of a Relationship-Entity-Datum Data Model.
Technical Report CSL 83-4, Xerox Corporation, Palo Alto Research Center, May, 1983.

[Chan 82] Chan, A., Danberg, S., Fox, S., Lin, W., Nori, A., and Ries, D.
Storage and Access Structures to Support a Semantic Data Model.
In *Proceedings of the Eighth International Conference on Very Large Data Bases*, pages 122-130. Mexico City, Mexico, September, 1982.

[Chen 76] Chen, P. P.
The Entity-Relationship Model: Toward a Unified View of Data.
ACM Transactions on Database Systems 1:9-36, 1976.

[Hammer 81] Hammer, M. and McLeod, D.
Database Description with SDM: A Semantic Database Model.
ACM Transactions on Database Systems 6(3):351-386, September, 1981.

[Herot 80] Herot, C. F.
Spatial Management of Data.
ACM Transactions on Database Systems 5:493-513, 1980.

[IBM 81] *Pascal Compiler*
First Edition edition, International Business Machines Corporation, Boca Raton, Fl., 1981.

[IBM 82] *Disk Operating System*
Second Edition edition, International Business Machines Corporation, Boca Raton, Fl., 1982.

[Kent 79] Kent, W.
Limitations of record-oriented information models.
ACM Transactions on Database Systems 4:107-131, 1979.

[King 82a] King, R.
A Unified Model and Methodology for Logical Database Design and Evolution.
PhD thesis, University of Southern California, May, 1982.

[King 82b] King, R. and McLeod, D.
The Event Database Specification Model.
In *Proceedings of International Conference on Improving Database Usability and Responsiveness*, pages 299-322. Jerusalem, Israel, June, 1982.

[Lyngbaek 83] Lyngbaek, P. and McLeod, D.
An Approach to Object Sharing in Distributed Database Systems.
In *Proceedings of the Ninth International Conference on Very Large Data Bases.* Florence, Italy, October, 1983.

[Lyngbaek 84] Lyngbaek, P. and McLeod, D.
Object Management in Distributed Office Information Systems.
USC Technical Report TR-84-301, University of Southern California, Los Angeles, Ca., February, 1984.

[McLeod 82] McLeod, D.
A Database Transaction Specification Methodology for End-Users.
Information Systems 7(3):253-264, 1982.

[Mylopoulos 80] Mylopoulos, J. and Wong, H. K. T.
Some features of the TAXIS data model.
In *Proceedings of the Sixth International Conference on Very Large Data Bases*, pages 399-410. 1980.

[Roussopoulos 75] Roussopolous, N. and Mylopoulos, J.
Using semantic networks for data base management.
In *Proceedings of the First International Conference on Very Large Data Bases*, pages 144-172. 1975.

[Shipman 81] Shipman, D.
The Functional Data Model and the Data Language DAPLEX.
ACM Transactions on Database Systems 2(3):140-173, March, 1981.

[Smith 77] Smith, J. M. and Smith, D. C. P.
Database Abstractions: Aggregation and Generalization.
ACM Transactions on Database Systems 2(2):105-133, June, 1977.

[Stonebraker 82] Stonebraker, M. and Kalash, J.
TIMBER: A Sophisticated Relation Browser.
In *Proceedings of the Eighth International Conference on Very Large Data Bases*, pages 1-10. Mexico City, Mexico, Sept. 8-10, 1982.

[Wells 84] Wells, R. P., Rochowansky, S., and Mellin, M. F. (editors).
The Book of IBM Software 1984.
The Book Company, Los Angeles, Ca., 1984.
ISBN 0-912993-02-2.

[Wong 82] Wong, H. K. T. and Kuo, I.
GUIDE: Graphical User Interface for Database Exploration.
In *Proceedings of the Eighth International Conference on Very Large Data Bases*, pages 22-32. Mexico City, Mexico, September, 1982.