An Object-Oriented Data Model for Database Modelling, Implementation and Access

LIPING ZHAO AND S. A. ROBERTS
Artificial Intelligence Group, Department of Computer Studies, University of Leeds, Leeds LS2 9JT

An Object-Oriented Data Model (OODM) has been defined for logical database design and database access. The OODM accommodates three types of relationship - aggregation, generalisation and particularisation - and provides four types of data operations for defining schema, creating database, retrieving objects and expanding objects. The expand operation is used in the context of previously retrieved objects or object types and allows automatic navigation through the database. The OODM supports a structured menu-based interface which allows a user to define the schema and retrieve from the database without any knowledge of the OODM. An important feature of this interface is that menu information is modelled as a set of interrelated objects stored in the database. Therefore menus can be created, accessed and updated in the same way as data. Furthermore, a basic set of menus is generated from the database schema and made available to the user for the purpose of answering simple queries. A prototype of the OODM has been implemented and used to model an application domain - Orchids Description and Classification.

Received June 1987, revised September 1987

1. INTRODUCTION

This paper presents an Object-Oriented Data Model (OODM) for database modelling, implementation and access. The purpose of the OODM is twofold: to provide a designer with an expressive tool for representing the information and to help a user to access the information. Towards this direction, the criteria of defining the OODM are: expressive ability, understandability, simplicity and accessibility.

In general, the OODM consists of two parts: the conceptual schema and data operations. The conceptual schema represents the real world as objects and relationships between these objects. Three types of relationships are identified: aggregation, generalisation and particularisation. Data operations include those for defining schema, creating database and manipulating objects. The components of this model are depicted in Fig. 1.

The concepts of aggregation and generalisation were first introduced by Smith and Smith.1,2 Generalisation refers to an abstraction which enables a class of individual objects to be thought of generally as a single named object, and aggregation refers to an abstraction in which a relationship between objects is regarded as a higher-level object. These two data abstractions have now been widely accepted in logical database design. In the OODM we introduce another type of relationship: particularisation. This relationship is very important when an object has a list of properties which would naturally be grouped into another object.

In order to increase its accessibility and to meet the aim of simplicity, the OODM provides a structured menu-based interface. This interface can be seen as an external model of the OODM which assists a user in accessing from a database and defining a database schema. An important feature of the interface is that menu information is partly stored in a menu database which can be manipulated in the same way as with objects, and partly generated automatically from the database and the database schema during the query time. Therefore, a single unchanging interface can support time-variable menus.

A prototype system of the OODM has been implemented using POP-11 under the UNIX operating system. This prototype currently provides four data operations: schema definition, database creation, data retrieval and data expansion. The menu-based interface, currently under development, supports five operations: schema definition, database creation, schema viewing, database browsing and data retrieval. A test database, which represents the domain of Orchids Description and Classification, has been created to demonstrate the usability of the prototype. This database is used for illustrating the various features of the OODM in this paper. In general, the example domain of Orchids Description and Classification includes species, variation and form of orchids. Each orchid is identified by its name and classified into a certain type of species, variation or form according to its field characters. Field characters of each orchid are characterised, for example, by leaf, perianth, fragrance and flowering period. Typical queries will include (1) retrieving all or specified species of orchids; (2) looking for all or specified field characters of a species; (3) finding all variations of one species; (4) according to field characters of an orchid, identifying its species.

The rest of the paper is organised as follows. In Section 2 a general account of related work is described. The conceptual schema of the OODM and its diagram representation are presented in Section 3 and data operations for the OODM are discussed in Section 4; Section 5 presents a menu-based interface to the OODM, followed by a brief description of the implementation of

![Figure 1. Components of the OODM.](https://academic.oup.com/comjnl/article-abstract/31/2/116/352255/)

116 THE COMPUTER JOURNAL, VOL. 31, NO. 2, 1988
2. BACKGROUND

Due to inadequacies and limitations of semantic expressiveness in traditional data models, the direction of research in data models has been shifted from traditional data models to semantic data models.\textsuperscript{7, 11, 13, 14} Representative of the semantic data models are: Entity-Relationship Model (E-R model) of Chen,\textsuperscript{9} Relational Model/Tasmania (RM/T) of Codd,\textsuperscript{9} Semantic Data Model (SDM) of McLeod and Hammer,\textsuperscript{5, 6} and Event Data Model of King and McLeod.\textsuperscript{10, 11} A comparison of these semantic data models can be found in Refs. 11 and 12.

A semantic data model is intended to provide user-understandable specification of data and to capture a substantial portion of meaning of the data (in order to convey it to users).\textsuperscript{7} The principal differences among data models are differences in the type of relationships that may be represented and in the constraints they impose on them. For example, the E-R model represents the information as entities and relationships between entities. However, in the E-R model the types of the relationships have not been distinguished. In the RM/T model, a type is represented as an entity relation (E-relation) which contains a single column specifying the 'surrogate' for every instance of the type. Attributes are represented by property relation (P-relation) which associates surrogate values with property values. Three types of relationships are identified in the RM/T: association (many-to-many relationship), designation (many-to-one relationship) and generalisation (type/subtype hierarchies). The SDM models a database as consisting of classes of objects. It accommodates the has-instance, has-attribute and has-subtype relationships. One of the special features of the SDM is its associated-user interface. The Event model represents a database as objects and application events (transactions), both of which are classified into object types. This model supports two types of relationship: generalisation – which is type and subtype relationship – and aggregation – which is formed via attributes and their inverses.

The ideas of these models have directed the development of the OODM.

3. CONCEPTUAL SCHEMA OF THE OODM

The basic constructs of the conceptual schema of the OODM are: objects, object types, attributes and relationships.

3.1 Object, object type and attribute

The definition of object, object type and attribute has been given by Chen.\textsuperscript{9}

An object is a real-world element or concept which can be distinctly identified. A person identified by his name (John Smith) and a course identified by title (Database Design) are examples of objects.

An object can be characterised by its properties and has an identifier. For instance, John Smith lives in St George Street or Database Design has module number D24. In this case, the strings 'John Smith' and 'Database Design' are identifiers of the objects introduced above: St George Street and D24 are properties for objects identified by John Smith and Database Design respectively.

It is interesting that while 'St George Street' has been described as a property of an object 'John Smith', it is of course providing an identifier of a street. Our model needs to be able to cope with this kind of dual role.

Objects are classified into different object types in terms of their properties; objects with the same properties (i.e. same meaning) are grouped into an object type which is identified by a type name.

The information about an object is expressed by a set of attribute-value pairs. John Smith and D24 are values. Values are classified into different value types, such as Name or Number.

An attribute can be formally defined as a function which maps from an object type into a value type or a Cartesian product of value type.

In the OODM there are three classes of attributes: value attribute, group attribute and aggregate attribute. Value attributes are proper attributes whose values are drawn from value types. Attributes chinesename and habitat...distribution of orchids are examples of value attributes. A group attribute can be seen as a representation of a group of attributes. For example, attribute field...characters in object type species is a group attribute for it represents a group of attributes [flower...number, flower...diam, flowering...period, fragrance,...., leaf,...., perianth, fruit]. The value of a group attribute is either a summarised description or a pointer to a set of values of a group of attributes depending on the retrieval context and the database extension. One object type can be referred to by the other object types through aggregate attributes. The value of an aggregate attribute is either an identifier of an object or a set of identifiers of objects of another object type, depending on the degree of the relationship.

An attribute may be defined as an identifier of an object, nonnull, single..valued or multi..valued. An identifier of an object must be nonnull; a single..valued attribute defines a one-to-one mapping from an object type to a value type; a multi..valued attribute defines a many-to-many (including one-to-many) mapping from an object type to a value type.

3.2 Relationships: aggregation, generalisation and particularisation

Objects may be related to one another via binary relationships. An object type can be regarded as a type of relationship which groups objects together. This type of relationship is called 'instance...of' relationship and is most essential: it exists inherently in object types. The relationships to be discussed belong to another category: they relate objects of different object types or objects of the same object type but in different roles.

Aggregation defines one-to-many or many-to-many relationships between object types. In the OODM, aggregation is a bi-directional relationship which allows two object types to refer to each other. The 'link' between two object types is an aggregate attribute pair – each associated with one object type. For instance, if we wish to model students and the courses taken by
them, an aggregate attribute course in object type student could be used to refer to object type course while an aggregate attribute student in object type course would refer to object type student. For the purpose of retrieval, object(s) of one object type can be traced from an object of another object type.

**Generalisation** defines type–subtype relationships between object types. Each subtype is a specialisation of its corresponding type. In the OODM, a generalisation is obtained by (i) grouping together objects of distinct object types to form another object type, or (ii) defining subtypes of an object type using predicates to constrain the values of attributes.

**Particularisation** defines one-to-one relationship between object types. Particularisation is a unidirectional relationship which allows an object (parent object) of one object type (parent type) to refer to an object (child object) of another type (child type) through a group attribute in the parent type, but not vice versa. A child object is a set of values of a group of attributes. This group of attributes can be represented by a group attribute in the parent type. Therefore, a child object represents properties of its parent object and its identifier is the same as its parent object.

The existence of a child object is determined by the value of the group attribute. If the value of a group attribute matches ‘see below’, there exists a child object, otherwise the child object does not exist. In the former case, the value of a group attribute may include a summarised description and then indicate ‘see below’ to allow the user to expand the detailed description in the child object. While in the latter case the value of a group attribute will simply include a text-like description and no further details can be expanded. Therefore, child objects are dependent objects as their existence depends on their parent objects.

### 3.3 Example: orchids database object type definitions

In Appendix 1 POP-11 syntax for defining object types for the domain of Orchid Description and Classification is shown. The description and classification in this example are based on the work of Pen and Liu and Turner Ettlinger. In this figure an example object type is species, with value attributes chinese_name, scientific_name, english_name, other_names and habitat and_distribution; group attribute field_characters and aggregate attributes variation and form. In the object type species, value attribute chinese_name serves as object identifier; group attribute field_characters has a group of attributes which are associated with an object type field_characters; aggregate attribute variation serves as a pointer to object type variation.

Observers may have noticed that, for example, group attribute ‘field_characters’ has its identical named object type ‘field_characters’, and aggregate attribute ‘variation’ has its identical named object type ‘variation’. The purpose of doing so is to provide a multi-level view of objects and object types. The point is that there is no rigid boundary between a type and an object. This is one of the distinctions between semantic data models and traditional data models.

An example of aggregation is the relationship between object types species and variation. These two object types can refer to each other through an aggregate attribute pair: ‘variation’ in species and ‘species’ in variation. The value of aggregate attribute variation is the object identifier(s) of variation; the value of aggregate attribute species is the object identifier of species. Therefore, object(s) of variation can be traced from a species object and vice versa. Aggregation between species and variation is a one-to-many relationship: a species may have more than one variation and a variation belongs to one species. The relationship between species and form and the relationship between variation and form are also aggregation relationships.

It is interesting to note that our domain illustrates a fundamental difference between the ‘semantic net’ used for knowledge representation and the semantic data model. If we were to store our information in a semantic net, each variation would become a kind of (ako) species. Although this might appear more natural it imposes restrictions on the data access paths, which is undesirable in the database context.

The particularisation relationship is introduced for capturing the semantics between an object type and its direct attribute and its indirect attributes.

For example, the group attribute field_characters of species can be characterised by a group of attributes [flower_number, flower_diam, flowering_period, fragrance, quality, leaf, perianth, fruit]. As this group of attributes characterises field_characters, they cannot be directly appended to other attributes of species without loss of semantics; on the other hand, this group of attributes has values which are properties of species. The meaning of the relationships between species and its (direct) attribute field_characters and its (indirect) attributes flower_number, flower_diam, ..., is represented by defining another object type field_characters which has a group of attributes and an identifier chinese_name from species. An object of species corresponds to at most one object in field_characters.

The relationship between species and perianth, the relationship between species and fruit and the relationship between field_characters and perianth are all examples of particularisation.

An example of generalisation is the relationship between species, variation and form. Since species, variation and form are specialised orchids, they can be grouped into a general object type orchid. Obviously, species, variation and form are specialisations of orchid, while orchid is generalisation of species, variation and form.

### 3.4 O-R diagram of the OODM

The development of the O-R diagram (Object-Relationship) of the OODM is influenced by the E-R model of Chen. Like the E-R model, the O-R diagram of the OODM is used for data analysis in logical database design.

In the O-R diagram, each object type is represented by a rectangular box. The aggregation relationship is represented by a dashed diamond-shaped box. The generalisation relationship is represented by a dashed triangle. The arrow represented the particularisation relationship. Fig. 2 illustrates the O-R diagram representation for the orchids domain.

For further demonstrating the O-R diagram we deliberately choose a well-known domain – ‘Suppliers-
Parts-Projects' database introduced by Date as an example. In order to illustrate the particularisation relationships, a new object 'address' has been added to the model. Fig. 3 shows its diagrammatic representation. With reference to Fig. 3, we can see certain differences between the E-R and OODM model.

First, in the E-R model, shipments and orders are treated as relationship types or composite entity types as introduced later by Chen, while in the OODM they are all object types. The reason is that shipments and orders are real-world concepts, which can be treated independently and can have their own properties. For example, 'quantities' and 'order-date' are properties of orders. A user may only be interested in shipments without being concerned about other information on suppliers and parts.

Secondly, in the E-R diagram relationships are represented by diamond-shaped boxes irrespective of the type of relationships, while in the O-R diagram relationships are classified as generalisation, aggregation and particularisation and represented by dashed triangle, dashed diamond-shaped box and arrow respectively. (For generalisation cf. Fig. 2.)

Finally, in the E-R model relationships are explicitly defined as relationship types, whilst in OODM relationships are expressed through attributes.

4. DATA OPERATIONS

Data operations for the OODM include schema definition, database creation, data retrieval, data expanding and data update. These operations (apart from data update) have been implemented using POP-11; their syntax is described in the following sections.

4.1 Schema definition and database creation

4.1.1 Schema definition

The operation for defining schema takes the form:

\[
\text{createobj}((\text{object\_type\_name}, [\text{attribute\_definition, relationship\_definition}])) \\
\rightarrow \text{object\_type\_name};
\]

The demonstration of this operation is shown in Appendix 1.

4.1.2 Database creation

The operation for creating database has the form:

\[
\text{addobj}((\text{object\_type\_name, object\_definition}));
\]

An example of this operation is given in Appendix 2.

4.2 Data retrieval

The variety of data retrieval operations depends on the schema of the data model. In the OODM two rudimentary data-retrieval operations for querying database are provided as follows.
4.2.1 Retrieve, project and select on an object type

The syntax of \texttt{retrieve} is:

\texttt{retrieve(object\_type\_name, list\_of\_attributes, list\_of\_conditions);}

where \texttt{list\_of\_attributes} has three patterns:

1. \texttt{[all]} --- projecting all the attributes in an object type;

2. \texttt{[all but att1 att2 ...]} --- projecting all the attributes with the exception of specified attributes: \texttt{att1, att2 ...};

3. \texttt{[att1 att2 att3 ...]} --- projecting on specified attributes: \texttt{att1, att2, att3 ...}

And \texttt{list\_of\_conditions} has the form:

1. \texttt{default};

In this case, retrieve becomes:

\texttt{retrieve(object\_type\_name, list\_of\_attributes);} .

2. \texttt{[att1, comparator value1 function att2 comparator value2 ...]}

Here \texttt{att1, att2} represent attributes which can be one of the following:

\texttt{word:} (e.g. xian\_ye\_chun\_lan, yes, no);
\texttt{number:} (e.g. 60, 50, 12.5);
\texttt{list:} (e.g. [found in Dalil, lian\_ban\_lan, xian\_ye\_chun\_lan]).

\texttt{Comparator} is one of:

- \texttt{= (equal to), < > (not equal to), > (greater than),
- \texttt{< (less than), >= (greater than or equal to),
- \texttt{<= (less than or equal to) includes}}

\texttt{Function} is one of

and (conjunction); or (disjunction).

The comparator \texttt{includes} is used to match a supplied value with one element of a list, for example:

\texttt{retrieve(species, [all], [variation includes fen\_hong\_duo\_xiang])};

retrieves the species which includes \texttt{fen\_hong\_duo\_xiang} among its variations.

Examples of using \texttt{retrieve} operation are given as below:

\texttt{retrieve(species, [all but variation form], 
[chinese\_name = chun\_lan]);

retrieve(variation, [all], [form = [jin\_huang\_duo\_xiang, ya\_huang\_duo\_xiang, huang\_hua\_duo\_xiang]]);

which retrieves the same variation as:

\texttt{retrieve(variation, [all], [form includes ya\_huang\_duo\_xiang]);

4.2.2 Expand

The operation \texttt{expand} takes the form:

\texttt{expand(object\_type\_name, list\_of\_attributes, list\_of\_conditions);}

where \texttt{list\_of\_attributes} and \texttt{list\_of\_conditions} have the same forms as those in operation \texttt{retrieve}.

The operation \texttt{expand} is designed as a sibling of \texttt{retrieve} and is used to support the aggregation and particularisation relationship. It is applied hand-in-hand with \texttt{retrieve}. For example, after retrieving an object or a set of objects from an object type, the user can use \texttt{expand} to present all the details of a group attribute in another object type or to trace object(s) of another object type through an aggregate attribute. In the former case, there exists a particularisation relationship between two object types. The latter case indicates that an aggregation relationship exists between two object types.

As an example of using this operation, consider the following queries:

1. \texttt{print english\_name, field\_characters, habitat\_and\_distribution of chun\_lan species};

2. \texttt{print all the information related to chun\_lan species.}

For the first query, we use \texttt{‘retrieve’}:

\texttt{retrieve(species, [english\_name, field\_characters habitat\_and\_distribution], [chinese\_name = chun\_lan]);

The second query will include querying field\_characters, perianth, fruit, variation and form of chun\_lan. We use \texttt{‘expand’} for this query.

\texttt{expand(field\_characters, [all]);

expand(perianth, [all]);

expand(fruit, [all]);

expand(variation, [all]);

expand(variation, [all], [chinese\_name = fen\_hong\_duo\_xiang]);

expand(form, [all]);

The above example shows us that \texttt{‘expand’} is an intelligent operation, for it can remember whatever \texttt{‘retrieve’} has done and then navigate between object types automatically. By using \texttt{‘expand’}, a user need not specify connection between object types and therefore can avoid the ‘connection trap’ as identified by Codd.\textsuperscript{8} Apart from navigating between object types, \texttt{‘expand’} can still navigate within the same object type. In other words, \texttt{‘expand’} has recursive (or reflexive) ability which can relate objects of the same type. This recursive feature is useful, for it can make the schema definition more natural and more sensible.

In the next section we shall see how this recursive feature has been used in defining object type of menu.

5. A MENU-BASED INTERFACE TO THE OODM

The interface is supported by a menu database and a Menu Management System (MMS) and currently provides five operations for defining schema, entering data, viewing schema, browsing data and retrieving data.

5.1 The menu, menu database and menu space

From the users’ viewpoint, a menu is a screen of information which consists of three sections: a menu title, a set of menu options and an interactive dialogue. The menu title provides a meaningful description of the menu’s purpose. The menu options represent the interface’s functions. The interactive dialogue appears in each menu to guide a user navigating through the menu network. In the menu-based interface of the OODM, menus are structured as a network which allows a menu to be reachable by more than one path and to be repeated by moving backward.

There are three classes of menu options in the menu-based interface of the OODM: predefined, user-defined and system-generated. The predefined menu options provide a primary set of functions of the interface and are domain-independent. For example, in our interface the primary functions will include: \texttt{‘Retrieve Data’}.
‘Define Schema’, ‘Browse Data’, ‘View Schema’, ‘Quit Menu’. The predefined menu options can be applied to any application domain represented by the OODM. The user-defined menu options are related to the particular application domain for answering the stored user queries. Each user-defined option corresponds to the translation of a user’s query into the data operations of the OODM.

These first two classes of menu option are stored in the menu database, each of which is represented as an object with an identifier and a menu path. All the objects in the menu database are grouped into an object type called a menu type. A menu path is either related to a set of menu options in the menu database or to a procedure which will generate menu options from the database schema and the database. These menu options, which are generated from the schema and the database, are called system-generated menu options, and are used to answer simple queries. The conceptual schema of the menus and a part of the menu database are given in Appendix 3.

A menu, then, can be defined as an instantiated menu type identified by a menu name. All the menus are generated automatically from the menu database, the database schema and the database, and structured as a network. The menu network can be defined as a menu space through which a user can navigate. Within this menu space, some sub-menu spaces (or sub-menu networks) can be formed dynamically during query time for answering different queries.

As an example, Fig. 4 illustrates a menu space generated from the menu database, the Orchids database schema and the Orchids database. The pre-defined menu options are shown in Table 2, the system-generated menu options are shown in Table 3, and the user-defined menu options are shown in Table 4.

5.2 The Menu Management System (MMS)

The objectives of the MMS are to generate menus, to display menus and to execute queries. To achieve these goals, the MMS should have the following functions: to control the movement of the menus and to generate menu network dynamically according to the user’s choice; to generate a basic set of menu options from the schema and the database and to access the other menu options from the menu database; to present menus, each with a title, a set of options and an interactive dialogue with user; to translate a user’s query into the data operations of the OODM and to execute these operations; to display the menu screens and the query results.

The MMS consists of a set of POP-11 procedures. Each procedure is responsible for one or part of the above functions. All these functions are initiated and controlled by a main procedure. A detailed description of the menu-based interface can be found in Technical Memorandum no. 6 by Zhao.90

5.3 Example: navigating through the menu-based interface

When the interface is initiated, the ‘main menu’ is presented as shown in Fig. 4. There are six functions available. ‘define schema’ presents a schema form which a user will fill in; ‘enter data’ is used to add data to the database; ‘view schema’ allows a user to explore the database schema; if a user wants to know what is available in the database before issuing queries, ‘browse

Figure 4. Automatically generated menu space.
data’ provides this convenience; ‘retrieve data’ allows a user to express queries in a piecemeal fashion on the basis of menu selections; ‘quit menu’ has its obvious meaning.

As an example, in Fig. 4, if a user wants to retrieve data from the orchids database, he/she need only specify the corresponding option. The ‘retrieve data’ menu is then returned with all the object types in the database. After the user selects an object, say species, another menu, called species, is presented with all the properties in species. After the user selects some properties for output and gives retrieval conditions, the results menu is generated. The menus visited by the user are ‘main menu’, ‘retrieve data’, ‘species’ and ‘retrieved data’, which are generated automatically according to the user’s choices and can be seen as a sub-menu space formed dynamically during the query time.

With the assistance of the menu-based interface, a user need not specify the connections between objects when retrieving data. Once an object is located, the other objects related will be automatically identified by the system and a user can navigate through a sub-menu space formed dynamically during the query time.

6. IMPLEMENTATION OF THE OODM PROTOTYPE

The programming language used to build the OODM prototype system is POP-11.19 As a general-purpose programming language, POP-11 has been used in AI research, VLSI design, graphics, text processing, compiler design, image processing, expert system design and interactive program development.18,19 POP-11 can also be used in the data-processing area, as it offers a large variety of the data structures such as lists, procedures and properties. Lists are a very general and useful basic structure for storing objects about the real world and relationships between objects. POP-11 provides a rich set of operations for manipulating lists, including list access, list construction and pattern matching. Pattern matching is a powerful programming technique which can be used for checking the correspondence of a list with a pattern. It is useful, for instance, for defining an interactive program whose responses depend on the pattern of words typed in. A property of POP-11 is a table of value associations. It functions as a kind of memory, and stores value associations according to their hashing code.

In our design, all the data operations of the OODM are defined as POP-11 procedures and supported by lists and lists operations; database is stored as POP-11 properties. We make use of POP-11 user stack and global variables to store information for the interaction of the operations and procedures, and make extensive use of pattern matching on lists for lexical analysis and retrieval. As an example, let’s see how the data operation retrieve works. The data operation retrieve takes three arguments. Its format is:

\[
\text{retrieve} (\text{object\_type\_name}, \text{list\_of\_attributes}, \text{list\_of\_conditions})
\]

list_of_attributes is a list of attributes to be projected, which has three patterns: [all], [all but att1 att2...], and [att1 att2...]. When a pattern is matched, it will initiate the corresponding operations.

list_of_conditions is a set of conditions used to select object(s). By using list access and list construction operations, this set of conditions is decomposed into four sublists: a list of attributes, a list of values, a list of comparators (‘=’, ‘<’, ‘>’, ‘...’) and a list of junctions (‘and’, ‘or’). According to these four lists, the system can work on the database using arithmetic comparators, logical conjunctions and disjunctions.

The implementations of the other operations are based on the same idea. This prototype is currently used on an experimental basis, to test the usability of the OODM. The domain of Orchids Description and Classification has been modelled using this prototype with encouraging results.

7. DISCUSSION

In this paper we have discussed the conceptual schema, the data operations and the menu-based interface of the OODM in detail. Examples have been given to illustrate the use of this model. The OODM has two advantages. First, it stores the knowledge of the connections between the objects, therefore the data operations become simple and meaningful. Secondly, its menu-based interface dynamically supports a menu space according to a user’s selections, which allows a user to traverse and explore both the meta data and the data of the database and to compose queries in a piecemeal fashion. The information flow between the OODM DBMS and the menu-based interface is depicted in Fig. 5.

The design of the OODM and its prototype is under the guidance of the four criteria. The OODM satisfies the first criterion – expressive ability, because its conceptual schema supports object type definition, attribute definition and relationship definition and accommodates three types of relationship: aggregation, generalisation and particularisation; also, a meta schema has been defined to represent the conceptual schema and the use of the OODM itself. The OODM also meets the second criterion – understandability, as the conceptual schema can be defined either using the data definition operation or under the directions of the menu-based interface. The third and fourth criteria – simplicity and accessibility – are also met, as the OODM provides both the data operations and the menu-based interface which accommodate both expert and novice users’ access to the data.

It is our intention to test the usability of our system by building personal databases for a small number of casual users. Feedback from this exercise will be used to influence future improvements.

![Figure 5. Information flow between modules.](image-url)
Currently the schema has to be written using POP-11 syntax; however, it would be an easy matter to write a ‘front-end’ Data Definition Language. Further improvement will include the implementation of data operations for generalisation and database update.

ACKNOWLEDGEMENTS

This work was completed using facilities in the Department of Computer Studies, University of Leeds. One of us (L. P. Z) is grateful to the Department of Computer Studies for financial support.

REFERENCES


APPENDIX 1. SCHEMA DEFINITION FOR ORCHIDS CLASSIFICATION DOMAIN

creatobj[Type: species
  [chinese_name from char_strings,single_value, identifier]
  [scientific_name from char_strings,single_value]
  [english_name from char_strings,single_value]
  [other_names from char_strings,multi_value]
  [field_characters from field_characters,single_value,group_att]
  [habitat_and_distribution from char_strings,multi_value]
  [variation from variation,multi_value,aggregate]
  [form from form,multi_value,aggregate]]
  -> species;

creatobj[Type: variation
  [chinese_name from char_strings,single_value,identifier]
  [scientific_name from char_strings,single_value]
  [english_name from char_strings,single_value]
  [other_names from char_strings,multi_value]
  [field_characters from field_characters,single_value,group_att]
  [habitat_and_distribution from char_strings,multi_value]
  [variation from variation,multi_value,aggregate]
  [form from form,multi_value,aggregate]]
  -> form;

creatobj[Type: field_characters
  [chinese_name from char_strings,single_value,identifier]
  [flower_number from char_strings,single_value]
  [flower_diam from char_strings,single_value]
  [flowering_period from char_strings,single_value]
  [habit_at_distribution from char_strings,single_value]
  [species from species,multi_value,aggregate]
  [form from form,multi_value,aggregate]]
APPENDIX 2. PART OF THE ORCHIDS DATABASE

addobj(species,,chun_lan,cymbidium_goeringii_rchb_f, spring_orchid,
[duo_duo_xiang,shuang_fei_yan(for 2 flowers/stalk)],
[see below],[found in kunming],
[fen_hong_duo_xiang,huang_hua_duo_xiang,bi_lu_duo_xiang,
bai_hua_duo_xiang,tao_hong_duo_xiang,cui_yu_duo_xiang],no);

addobj(variation,[fen_hong_duo_xiang,
cymbidium_goeringii_rchb_f_var_goeringii_pen_et_liu,
no,no,[see below],[found in most parts of yunnan],
chun_lan,no]);

addobj(variation,[huang_hua_duo_xiang,
cymbidium_goeringii_rchb_f_var_armeniacium_pen_et_liu,
no,no,[see below],[found in wenshan,fuyung,kunming],
chun_lan,[jin_huang_duo_xiang,ya_huang_duo_xiang,huang_hua_su_xin])];

APPENDIX 3. SCHEMA DEFINITION FOR MENUS AND PART OF MENU DATABASE

vars menu;
createobj([Type: menu

addobj(menu,[quit, option, [Quit Menu], quit, no]);
addobj(menu, [ret, option, [Retrieve Data], no, spobj]);
addobj(menu, [def, option, [Define Schema], no, defobj]);
addobj(menu, [ent, option, [Enter Data], no, entdata]);
addobj(menu, [view, option, [View Schema], no, viewsch]);
addobj(menu, [brow, option, [Browse Data], no, browdata]);