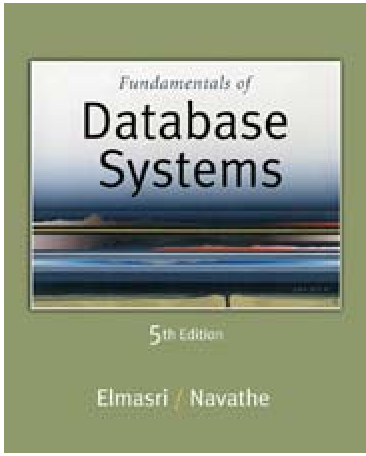
Chapter 2

ODMG Standard: Languages, and Design



The ODMG Standard

1 Overview of the Object Model ODMG

2 The Object Definition Language DDL

3 The Object Query Language OQL

4 Object Database Conceptual Model

5 Summary

The Object Model of ODMG

- Provides a standard model for object databases
- Supports object definition via ODL
- Supports object querying via OQL
- Supports a variety of data types and type constructors

The Main Class Hierarchy

- Denotable_Object
  - Object
    - Atomic_Object
    - Structured_Object
  - Litteral
    - Atomic_Literal
    - Structured_Literal
- Characteristics
  - Operation
  - Relationship

ODMG Objects and Literals

- An object has four characteristics
  1. Identifier: unique system-wide identifier
  2. Name: unique within a particular database and/or program; it is optional
  3. Lifetime: persistent vs transient
      1. coterminus_with_procedure
      2. coterminus_with_process
      3. coterminus_with_database
  4. Structure: specifies how object is constructed by the type constructor and whether it is an *atomic* object
- Main attributes of an object:
      1. has_name? : Boolean
      2. names : Set<String>
      3. type : Type
- Main Operations:
      1. delete()
      2. same_as(OID : Object_id) : Boolean

ODMG Literals

- A literal has a current value but not an identifier
- Two types of literals
  1. *atomic*: predefined; basic data type values (e.g., `short, float, boolean, char`)
  2. *structured*: values that are constructed by type constructors :
     1. Immutable_ Structure (Date, Time, Timestamp, Interval)
     2. Immutable_Collection (Bit_String, Character_String, Enumeration)

Object Definition Language

- ODL supports semantics constructs of ODMG
- ODL is independent of any programming language
- ODL is used to create object specification (classes and interfaces)
- ODL is not used for database manipulation

State modeling

- Attributes
    - ***attribute*** dataTypeName attrName;
- Relations
    - ***relationship*** dataTypeName relName ***inverse*** referencedClassName::relaNameInv;
- Operation
- State modeling is defined by an Interface

Interface and Class Definition

- ODMG supports two concepts for specifying object types:
  - **Interface**
  - **Class**
- There are similarities and differences between interfaces and classes
- Both have behaviors (operations) and state (attributes and relationships)

ODMG Interface

- An interface is a specification of the abstract behavior of an object type
  - State properties of an interface (i.e., its attributes and relationships) cannot be inherited from
  - Objects cannot be instantiated from an interface

ODMG Class

- A class is a specification of abstract behavior and **state** of an object type
  - A class is **Instantiable**
  - **Supports “extends”** inheritance to allow both state and behavior inheritance among classes
  - **Multiple inheritance** via “extends” is not allowed

ODMG Interface Definition: An Example

- Note: interface is ODMG's keyword for class/type

```
interface Date:Object {
  enum weekday{sun,mon,tue,wed,thu,fri,sat};
  enum Month{jan,feb,mar,…,dec};
  unsigned short year();
  unsigned short month();
  unsigned short day();
  …
  boolean is_equal(in Date other_date);
};
```

Built-in Interfaces for Collection Objects

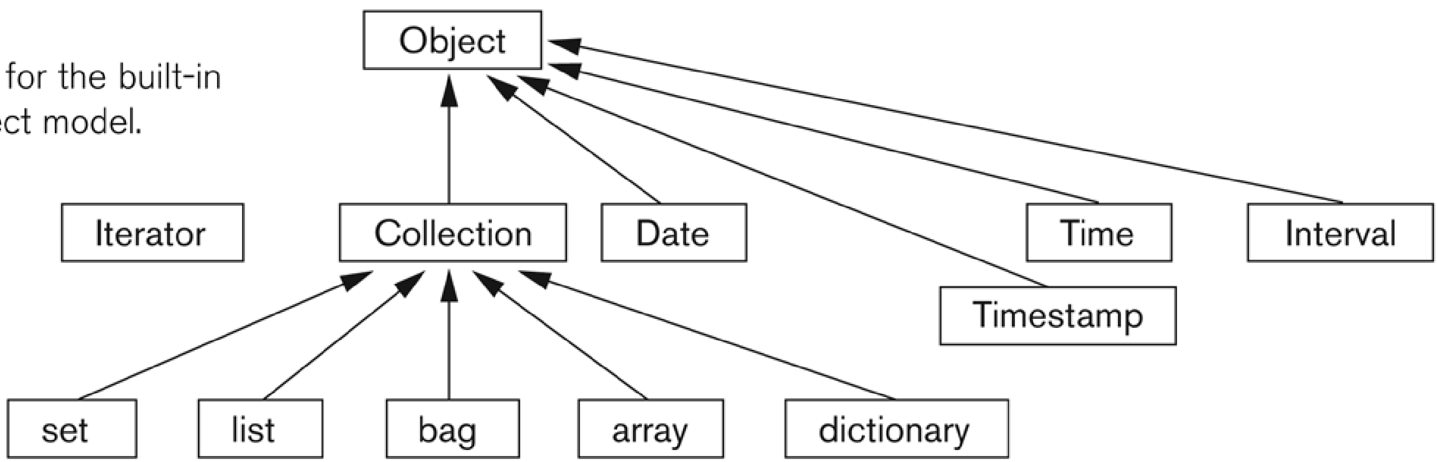
- A **`collection`** object inherits the basic **`collection`** interface, for example:
  - **`cardinality()`**
  - **`is_empty()`**
  - **`insert_element()`**
  - **`remove_element()`**
  - **`contains_element()`**
  - **`create_iterator()`**

Collection Types

- Collection objects are further specialized into types like a set, list, bag, array, and dictionary
- Each collection type may provide additional interfaces, for example, a set provides:
  - **`create_union()`**
  - **`create_difference()`**
  - **`is_subset_of(`**
  - **`is_superset_of()`**
  - **`is_proper_subset_of()`**

Object Inheritance Hierarchy

Figure 21.2
Inheritance hierarchy for the built-in interfaces of the object model.



Atomic Objects

- **Atomic objects** are user-defined objects and are defined via keyword **`class`**
- An example:

```
class Employee (extent all_emplyees key ssn) {
   attribute string name;
   attribute string ssn;
   attribute short age;
   relationship Dept works_for;
   void reassign(in string new_name);
}
```

Class Extents

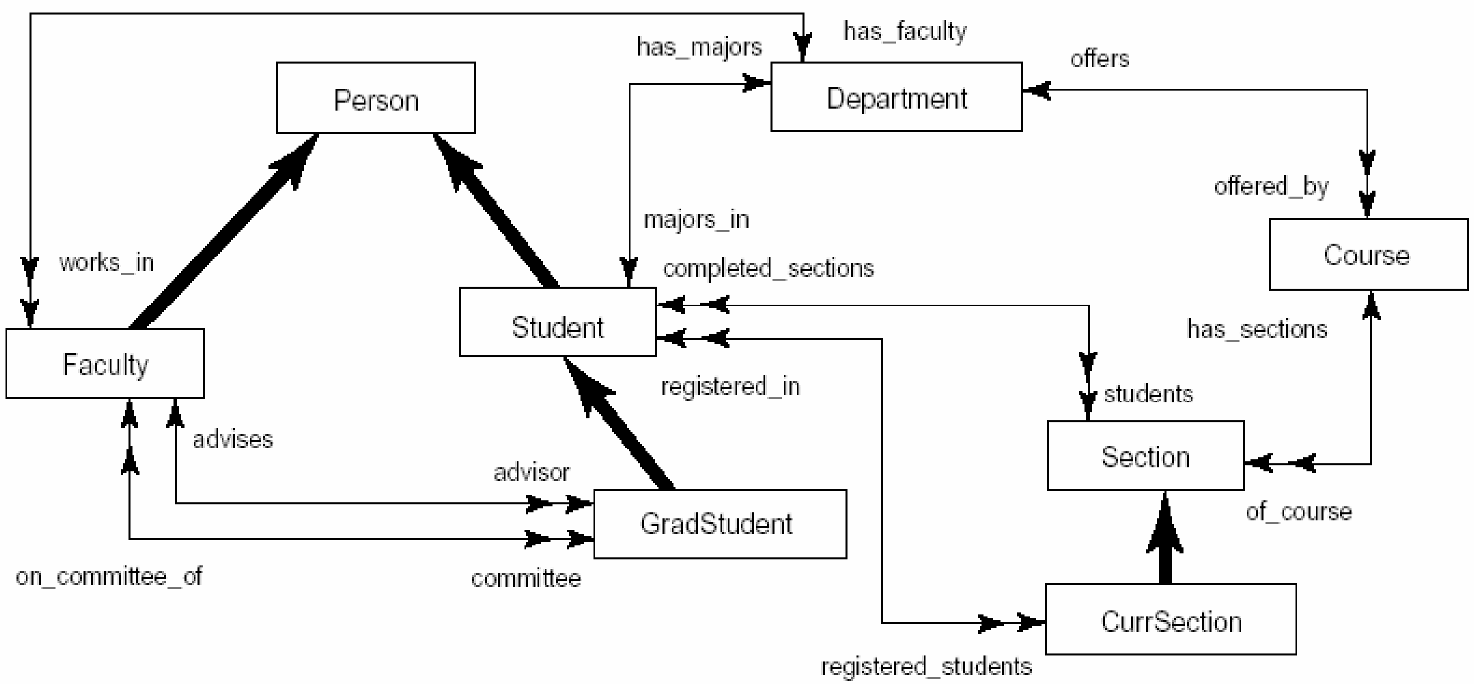
- An ODMG object can have an **`extent`** defined via a class declaration
  - Each `extent` is given a name and will contain all persistent objects of that class
  - For `Employee` class, for example, the `extent` is called `all_employees`
  - This is similar to creating an object of type `Set<Employee>` and making it persistent

Class Key

- A class key consists of one or more unique attributes
- For the `Employee` class, the key is `ssn`
  - Thus each employee is expected to have a unique `ssn`
- Keys can be composite, e.g.,
  - `(`**`key`**` dnumber, dname)`

ODMG Object Model (Cont.)

- ODL --- a database schema specification language
- Example



ODL Examples
A Class With Key and Extent

- A class definition with “extent”, “key”, and more elaborate attributes; still relatively straightforward

```
class Person (extent persons key ssn) {
  attribute struct Pname {string fname …} name;
  attribute string ssn;
  attribute date birthdate;
  …
  short age();
}
```

ODL Examples (2)
A Class With Relationships

- Note extends (inheritance) relationship
- Also note “inverse” relationship

```
class Faculty extends Person (extent faculty) {
  attribute string rank;
  attribute float  salary;
  attribute string phone;
  …
  relationship Dept works_in inverse
  Dept::has_faculty;
  relationship set<GradStu> advises inverse
  GradStu::advisor;
  void give_raise (in float raise);
  void promote (in string new_rank);
};
```

Inheritance via “:” – An Example

```
interface Shape {
  attribute struct point {…} reference_point;
  float perimeter ();
  …
};

class Triangle: Shape (extent triangles) {
  attribute short side_1;
  attribute short side_2;
  …
};
```

Object Query Language

- OQL is DMG's query language
- OQL works closely with programming languages such as C++
- Embedded OQL statements return objects that are compatible with the type system of the host language
- OQL's syntax is similar to SQL with additional features for objects

Simple OQL Queries

- Basic syntax: select…from…where…
  - SELECT d.name
  - FROM d in departments
  - WHERE d.college = ‘Engineering’;
- An **entry point** to the database is needed for each query
- An `extent` name (e.g., departments in the above example) may serve as an entry point

Iterator Variables

- Iterator variables are defined whenever a collection is referenced in an OQL query
- Iterator d in the previous example serves as an iterator and ranges over each object in the collection
- Syntactical options for specifying an iterator:
  - `d in departments`
  - `departments d`
  - `departments as d`

Data Type of Query Results

- The data type of a query result can be any type defined in the ODMG model
- A query does not have to follow the `select…from…where…` format
- A persistent name on its own can serve as a query whose result is a reference to the persistent object. For example,
  - departments; whose type is set<Departments>

Path Expressions

- A **path expression** is used to specify a path to attributes and objects in an entry point
- A path expression starts at a persistent object name (or its iterator variable)
- The name will be followed by zero or more dot connected relationship or attribute names
  - E.g., departments.chair;

Views as Named Objects

- The **define** keyword in OQL is used to specify an identifier for a **named query**
- The name should be unique; if not, the results will replace an existing named query
- Once a query definition is created, it will persist until deleted or redefined
- A view definition can include parameters

An Example of OQL View

- A view to include students in a department who have a minor:

```
define has_minor(dept_name) as
select    s
from      s in students
where     s.minor_in.dname=dept_name
```

- has_minor can now be used in queries

Single Elements from Collections

- An OQL query returns a collection
- OQL's `element` operator can be used to return a single element from a singleton collection that contains one element:

```
element (select d from d in departments
where d.dname = 'Software Engineering');
```

- If d is empty or has more than one elements, an **exception** is raised

Collection Operators

- OQL supports a number of aggregate operators that can be applied to query results
- The aggregate operators and operate over a collection and include
  - `min, max, count, sum, avg`
- `count` returns an integer; others return the same type as the collection type

An Example of an OQL Aggregate Operator

- To compute the average GPA of all seniors majoring in Business:

```
avg (select s.gpa from s in students
  where s.class = 'senior' and
  s.majors_in.dname ='Business');
```

Membership and Quantification

- OQL provides membership and quantification operators:
  - (e **`in`** c) is true if e is in the collection c
  - (**`for all`** `e` **`in`** `c:` b) is true if all e elements of collection c satisfy b
  - (**`exists`** `e` **`in`** `c:` b) is true if at least one e in collection c satisfies b

An Example of Membership

- To retrieve the names of all students who completed CS101:

```
select s.name.fname s.name.lname
from   s in students
where  'CS101' in
  (select c.of_course.name
  from c in s.completed_sections);
```

Ordered Collections

- Collections that are lists or arrays allow retrieving their **first** and **last** elements
- OQL provides additional operators for extracting a sub-collection and concatenating two lists
- OQL also provides operators for ordering the results

An Example of Ordered Operation

- To retrieve the last name of the faculty member who earns the highest salary:

```
first (select struct
        (faculty: f.name.lastname,
              salary: f.salary)
from f in faculty
ordered by f.salary desc);
```

Grouping Operator

- OQL also supports a grouping operator called **`group by`**
- To retrieve average GPA of majors in each department having >100 majors:

```
select deptname, avg_gpa:
  avg (select p.s.gpa from p in partition)
from s in students
group by deptname: s.majors_in.dname
having count (partition) > 100
```

Summary

- Proposed standards for object databases presented
- Various constructs and built-in types of the ODMG model presented
- ODL and OQL languages were presented