Chapter 12 – Object Database Standards, Languages and Design

See Ch 11 for Object-Oriented Concepts (much is review from typical OO Programming)

ODMG Model

- Object Data Management Group
- data model on which ODL and OQL are based
- provides standard data model for object-oriented databases
- Objects and Literals
    - Basic building blocks of the object model
    - Object has both object identifier and state (current value)
    - Value can have complex structure
    - state can change over time by modifying value(s)
    - Object describe by 4 main characteristics:
        - Identifier – unique system wide identifier
        - Name – unique name within database – sometimes used as entry points to database
        - Lifetime – persistent (database object) or transient (program object)
        - Structure – how the object is constructed – atomic or collection object
    - Literals – 3 types
        - Atomic – values of basic data types – long, short, unsigned, etc.
        - Structured – constructed like a C++ struct
            - Ex: pre-defined structured type Timestamp

```
Interface Timestamp : Object {
  unsigned short year();
  unsigned short month();
  unsigned short day();
  unsigned short hour();
  unsigned short minute();
  unsigned short second();
  unsigned short millisecond();
  Timestamp plus(in Interval some_Interval);
  Timestamp minus(in Interval some_Interval);
  boolean is_equal(in Time other_time);
  boolean is_greater(in Time other_time);
}
```

            - ODMG keyword interface for type or class

  - Collection – specifies a value that is a collection of objects or values
    - Collection does not have an object id – members do
    - Set, bag, list,array, dictionary (look-up table)
    - Built-in operations:
      - Is_empty(), insert_element(e), remove_element(e), contains_element(e)
      - Create_iterator() – creates iterator object that can iterate over each element in collection
        - Reset() – resets iterator at first element of collection
        - Next_position(), get_element()
    - More about specific types of collection objects in book

  - All objects in the ODMG object model inherit the basic interface Object
    - basic operations are inherited by all objects – ex:
      - copy – creates new copy of object
      - delete
      - same_as - compares to another object
    - operations applied using dot notation – myObject.same_as(p)
    - type inheritance – uses colon notation

- Atomic (user-defined objects)

Example: Parts/Suppliers database from EER handout:

```
Class Product
(  extent all_products
   key prod_num)
{
   attribute string  name;
   attribute string  prod_num;
   attribute string  description;
   attribute date    date_produced;
   attribute enum Color{red, blue, green, yellow} color;
   attribute set struct Parts {
                                int quantity,
                                Part part
                              } parts;
   relationship Purchased Item  is_for
```

```
                  inverse PurchasedItem::is_for;
   void add_product(string name, string prod_num)
                  raises (prod_name_not_valid);
   void add_part(Part new_part);
}
```

- use keyword class
- any user-defined object that is not a collection is an atomic object
- ex: In a parts-suppliers database (see EER handout) – specify object type for Product object
- 3 parts of a user-defined object - attributes, relationships and operations
  - attribute – property that describes some aspect of an object
    - have values – literals either simple or complex
    - can also be object-ids of other objects
    - ex: prod_num – simple; parts – complex and other objects
  - relationship – property that specifies two objects in DB are related to each other
    - only binary relationships
    - pair of inverse references via keyword relationship
    - some relationships (ER type) are modeled as an attribute in an object (ex: parts)
  - operations – specify behavior of the object
    - specify names of exceptions that can occur during operation execution

- Interfaces and Classes
  - interface – specification of abstract behavior of an object
    - specifies operation signatures
    - non-instantiable
    - used for specifying abstract operations that will be inherited by classes or other interfaces
    - behavior inheritance – specified with ":" symbol
  - class – specification of abstract behavior and abstract state of an object state
    - instantiable
    - behavior and state inheritance – uses "extends" keyword
    - supertype and subtype are classes

- multiple inheritance not allowed with “extends”
- can have multiple inheritance by inheriting any number of interfaces, and at most one class

- Extents
  - set object that holds all persistent objects of the class
  - enforces set/subset relationship between extents of superclass and its subclasses

- Keys
  - key consists of one or more properties (attributes or relationships) whose values are constrained to be unique for each object in the extent
  - composite key – made up of several properties

- Factory Object
  - generate or create individual objects via its operations
  - interface ObjectFactory – single operation new()
  - user-defined objects can inherit this interface to become factory objects
  - provides constructor operations for new objects

```
class ProductFactory : FactoryObject {
  ...
}
```

- Database
  - interface DatabaseFactory – to create new database objects
  - interface Database
    - has own name
    - bind operation to assign unique names to persistent objects in a database
    - lookup – returns object with specified name
    - unbind – removes name from database

- Object Definition Language – ODL
    - independent of any programming language
    - used to create object specifications
    - example above is in ODL notation
    - several possible mappings from an object schema diagram (ER or EER) into ODL classes
    - entity types mapped to ODL classes
    - inheritance done using extends
    - no direct way to map unions or do multiple inheritance
        - read chapter for more details of mapping

- Object Query Language – OQL
    - syntax similar to SQL – with extensions for ODMG concepts
    - designed to work closely with languages which have a ODMG binding
    - some sample queries here – we won’t have time for too much detail

- Sample queries

```
Q0:     SELECT P.NAME
        FROM P IN PRODUCTS
        WHERE P.COLOR = “BLUE”
```

- entry point to database needed for a query – any named persistent object
    - usually the name of the extent of the class

- iterator variable – P in example –
- type of result – bag<string> since we are selecting P.NAME
- in general – result of a query is a bag for select ... from   set for select distinct ... from

```
Q1:     products
```

- any persistent name is a query – result is a reference to that object
- Q1 returns reference to a collection of all persistent product objects
- if we give a particular product object a name – “widget” – through bind – we could do the following

Q1a: widget

- this would return a reference to the object

- Once an entry point is specified – path expression can be used to specify a path to related attributes

Q2: widget.color

Q2a: widget.parts

Q2b: widget.is_for

- can specify a query that results in a complex structure using struct keyword

Q3: order123.customer.custname

Q3a: select struct(custname:struct(last_name:c.name.lname, first_name:c.name.fname))

from c in order123.customer

- retrieves the name from the customer of order123

- Specifying views as named queries

V1: define colored_products(color) as

SELECT P

FROM P IN PRODUCTS

```
        WHERE P.COLOR = color
```

- can write a query: colored_products(“green”)

- can select single elements from collections:

```
Q4: element ( select p
                from p in products
                where p.name = “widget5”)
```

- guaranteed to return a single element – if more than one is in the result, exception is raised

- aggregate functions and quantifiers

```
Q5: count ( p in colored_products(“blue”))

Q6: avg ( pi.quantity
    from pi in PurchasedItem
    where pi.is_for.color = “blue”)
```

- membership condition

```
Q7: select c.custname.lname, c.name.fname
    from c in customer
    where “blue” in
            (select p.name
            from p in c.orders.consists_of.is_for)
```

-
  -

-