A Short Introduction to O2, O2C and OQL

November 3, 2000

OODB Basics

An OODB is a database system (provides persistance, transaction support, query evaluation) which supports a rich type system.
In contrast, tables in a relational DB have very restrictive structure : they consist of tuples of the built-in types.

The O2 System

The O2 system provides for easy development of object oriented database applications. It consists of :

- O2 engine and O2 store, which provides for persistance, OQL query evaluation, transactions, logging/recovery.
- A custom API in the form of a programming language O2C.
- Interfaces with C++, Java, Relational DBs
- other stuff

O2C : Intro

- O2C is an object oriented high level programming language.
- It is a 'superset' of C, and is the language of the O2 shell.
- It allows for the quick development of applications using the O2 system.
- It is the language used for specifying the body of the methods for the classes used in a database instance.

O2C : Basic Types

- integer
- string
- boolean
- real
- char
- bits

O2C : Constructed Types

1. tuple(a_1 : type1, . . ., a_k : typek)

2. set(type)

3. unique set(type)

4. list(type)

5. named types

```
type address : tuple ( street : string,
      city : string, state : string,
      zip : integer ) ;
```

named types must be non-recursive.

O2C : Classes

O2C allows the user to define classes. A class has a value (usually a tuple type, fields of this tuple are the attributes (data) of the class), and methods.

Accessibility of data/methods can be of 3 types : public, private (default) or read.

Class Specification

```
class Person
public type tuple (
  name: string,  /* data members */
  age: integer
)
method    /* method declarations */
   public init (n: string, a: integer),
   public print,
   public get_age : integer
end;
```

O2C : Inheritance

```
class Employee inherit Person
public type tuple( id : integer, salary : integer,
                   department : string )
end ;
```

Employee inherits attributes, methods from Person.

O2C : Multiple Inheritance

Multiple inheritance is supported in O2C.

```
class Student inherit Person
public type tuple( id : integer,
            courses : set(
                      tuple( course_id : string,
                         title : string, fee : integer) ) )
method public total_fee : integer
end ;
```

```
class StudentEmployee inherit Student, Employee
rename attribute id from class Student as student_id,
attribute id from class Employee as employee_id
end ;
```

O2C : Operators, Expressions

- Set Operators :

$+$	set union
$-$	set difference
$*$	set intersection
in	set membership

- Boolean Operators : $==, !=, >, >=$ etc.
  In case of objects, $==, !=$ tests for object equality (like in Java) and inequality respectively.

- Dereferencing objects : $*p$ gives the value of object p.

O2 schema

An O2 schema consists of

- Class definitions
- Persistent Name declarations : These can be thought of as global variables which are persistent and visible to all methods, application programs using this schema.

```
name people : set(Person) ;
```

O2 Base

- An O2 Base contains the persistent objects belonging to the classes in the associated schema.

- A base is always associated with a schema.

- A schema may have multiple bases

Object persistence

An object instance is persistent if

- it is declared with the `name` keyword.
- it is referenced (reachable from) another persistent object.

O2C : Method Implementations

```
method body getID : integer in class Student {

  o2 integer i ;
  o2 tuple( n : string, i : integer )  tp ;
  o2 Person p ;

  return self->id ;
} ;
```

The keyword `o2` is used to declare temporary (local) variables.

Special Methods

- init : this acts as the constructor.
  this method is not inherited. If required, superclass constructors need to be called explicitly.
  all classes have a default constructor, which initialises all attributes to their default values.

```
method body init( n : string, a : integer ) in class Student
{
    self->Person@init( n, a ) ;
} ;
```

O2C : Control Flow

- All the C constructs are applicable.
- `for ( x in collection ) { ... }`
  This construct allows iteration over the elements of a set. (or list)

Creating a Schema

```
create schema testschema ;
set schema testschema ;

 class definitions
confirm classes ;
set base myDB ;

run body {
 O2C code
}
```

O2 Applications

An application is a set of related procedures (programs or transactions) which operate on the database instance. (the current schema and base)

```
application EnrollmentManager
   program public addclass(
     student_name : string, class : string ) : boolean,
   program public dropclass(
     student_name : string, class : string ) : boolean
end ;
```

O2 Programs

```
transaction body addclass(
  student_name : string, class : string ) : boolean
      in application EnrollmentManager
{
   o2 Student x ;
   for ( x in Students ) {
     if( x->name == student_name ) {
     }
   }
} ;
```

OQL

OQL is the standard query language for OODBs.

OQL is a superset of SQL. OQL queries have the form

```
SELECT result_tuple_expr
FROM collection_expression
(WHERE condition
 GROUP BY group_by_attrs
 HAVING condition
 ORDER BY order_by_attrs
)
```

More OQL

- List all student names and their id's

```
SELECT sname : s.name, sid : s.id
FROM s in Student
```

- List John's courses

```
SELECT cid : c.id, cname : c.title
FROM Student s, s.courses c
WHERE s.name = "John"
```

More OQL

- List students taking a class taught by Prof. Smith

```
SELECT sname : s.name \\
FROM Student s, s.courses c \\
  Faculty f, f.teaches t \\
WHERE c.id = t.id AND f.name = "Smith" \\
```

OQL : Group By

- Find the set of students enrolled in each course

```
SELECT struct( cid : c.id,
          students : (SELECT p.s.name, p.s.id
                      FROM partition p)
       )
FROM Student s, s.courses c
GROUP BY c.id
```

Example DB

```
Student  ( Name     Courses (cid) )
---------------------------
s1         Bob     (10, 11, 12)
s2         Bill    (11, 12, 14)
s3         John    (10, 14)
s4         Ann     (10, 11, 12)
s5         Lisa    (11, 14, 15)
```

Group By : Intermediate Collection

```
tuple( cid : 10,
   partition : set( tuple( s : s1, c : tuple( cid : 10 ) ),
                    tuple( s : s3, c : tuple( cid : 10 ) ),
    tuple( s : s4, c : tuple( cid : 10 ) )
  )
      ),
tuple( cid : 11,
     partition :

  )
```

OQL : Aggregate functions

- Find the number of students enrolled in each course

```
SELECT struct( cid : c.id,
           enrollment : count ( SELECT p.name, p.id
                                FROM partition p )
         )
FROM Student s, s.courses c
GROUP BY c.id
```

OQL : Aggregate functions

- Find employees earning more than twice the average salary.

```
SELECT ename : e.name
FROM Employees e
WHERE e.salary > 2*AVG( SELECT e1.salary FROM Employees e1 )
```

OQL : Set Operations

The operators for set union , intersection and difference are `UNION`, `INTERSECT` and `EXCEPT` respectively.

- List the courses taken either by John or Bob

```
SELECT c.id, c.title
FROM Student s, s.courses c
WHERE s.name = "John"
UNION
SELECT c.id, c.title
FROM Student s, s.courses c
WHERE s.name = "Bob"
```

OQL : Set Membership

OQL has the operators `IN` and `EXISTS`.

- List students who are not employees.

```
Select s.name
FROM Student s
WHERE NOT (s.name IN ( SELECT e.name
                       FROM Employees e ) )
```

Executing Queries in O2

- query shell
- embedded queries in O2C code
  `o2query( result, querystring ) ;`

References

- The official O2 documentation is available in the directory `/software/nonrdist/o2_5.0/manuals`
  The O2C beginners guide is `o2cbegui.pdf`.
  The O2C Reference is `o2cref.pdf`.

- The class web page has a tutorial on O2C and OQL.