3 Object Oriented Database Systems

3.1 Introduction
3.2 The ODMG data model
3.3 OQL in a nutshell
3.4 Persistence Architectures
3.5 The Object Protocol Model

OO DB model: see Kemper / Eickler;
OPM : papers 1,2

Data models: the spectrum

- Databases
  - Rigid data models: relational, object-oriented
  - Database conformant to schema
  - Semantics of query q: subset of database
  - No texts, images, ... (originally)
- Semi structured DB / XML
  - Schema more flexible – if any
  - Many schema items
  - Text plays a big role
  - Semantics of queries: substructure of DB
- Information Retrieval
  - Data model: objects are sequences of terms
  - No modeling restrictions (natural language!?)
  - Semantics of query q: DB entries ordered by similarity to q (Ranking)
- Natural Language
  - VERY difficult to process automatically
  - not really an option

More structure

We are here today

Less structure
3.1 Introduction: Why OO Databases

• Data modeling: What is appropriate for bio data?
  – No clear answer…
  – flexibility, ease of use, few restrictions….
• Object orientation: much more flexibility than relational systems
• The essence of Object orientation
  – data abstraction
  – inheritance
• OO: a programming language paradigm … adapted for databases

Introduction: OO principles

• Abstraction in OO Languages and DB
  – hide representation, encapsulation
  – use interface operations
    class Angel {
      private double angel; // radians
      public double getDegree() {
        return java.lang.math.PI * angel /180;
      }
      public double getRad(){return angel;}
      public void set(double val){…}
      public void setDegree(double val) { … }
    }
  – Operation interface not forced by most PL or OODB
  – Database terminology: "behavior" of entities if they define operations
Introduction: OO principles

• Inheritance
  – Type extension / Specialization / Generalization
    ```java
class publication {
    private String title;
    private String[] authors;
    public print();
    ...
}
class book extends publication {
    private String isbn;
    private int printing;...
    public print(){...} ...
}
```
  – type polymorphism and dynamic binding
    ```java
(set_of publication) mySet; ...
for each d in mySet { d.print(); ...}
```

Motivation for Objects in databases

What is missing in the Relational Model?
• Type constructors
  – Tuple types
  – Compound structures, recursive structures
    ```java
class Project {
    Set_Of Sequence sequences;
    Project project_parent;...
}
```

• Object Identity
  - Identification of "objects" in RDB by unique value
  - May change over time?
  - Frequently artificial keys as a substitute
Motivation for Objects

- Data abstraction
  - Encapsulation of state
  - Hiding of representation
  - "Behaviour"
- Type extensions and inheritance
  - Part of the EER Model
  - Emulated by relations
  - Type versus extent?

Object oriented Programming Languages...

- ... have all the desired features
- "Impedance mismatch"
  - View of data and operations on data in DBS and in PL differs heavily

- Missing: persistence, concurrency, indexing fault tolerance, query language
Object oriented Programming Environment

Persistent PLang as a substitute for DBS?

- Data model tightly bound to OO paradigm
  - … but data live independent of any programming model
  - Data model should be suited for all kinds of programming models
  - Simplistic RDM a good compromise?

- No common OO data model
  - Eiffel, Java, C++, C#, Smalltalk,…
  - Standardization needed!

- Object Data Management Group (ODMG) *
  data model (1993)

- Competing approach: extension of the RDM by OO concepts

* Different from Object Management Group (OMG) for Corba standardization

3.2 The ODMG Data Model

Environment of the OO Database Model

![Diagram showing compatibility between object-oriented models and languages]

Compatibility required: oodb Model and oo language models
Defined bindings: C++, Java, Smalltalk, C#

drawing by FJ Kemper
ODMG Type System

• Types
  – Atomic literal types
    • Primitive types
      integer, float, character, char strings, boolean, enum
  – Collection literal types
    • set, bag
    • list, array
    • dictionary
  – structures (record, C / C# struct)

```
struct conditions
{
  float humidity; float pressure; float temperature;
};
set <enum strategy_type (shotgun, primer, del_clone, genomic, transposon)> strategy_Set;
```

• Class Types
  – Define "state and behaviour" of objects of this type
  – Behaviour: operations (methods, functions)
  – State: attributes and relationships

• Class types can be instantiated: new...

```
class Experiment {
  attribute struct {float humidity; float pressure; float temperature;} cond;
  attribute time t;
  attribute People experimenter; // an object
  attribute set <enum ExType (ALPHA, BETA, PI)> experimentType
}
```
ODMG Data Model

• Relationships
  – Associate one object of class C1 to one or more of class C2
    • Only binary relationships, recursive relationships (C1 == C2) allowed
      
      ```
      relationship People experimenter
      inverse Experiment::makes;
      ```

  • Relationships preserve integrity ("foreign keys")
  • in contrast to attributes: attribute People experimenter;
  – 1:1, 1:N, M:N relationships
  – implementation of relationships encapsulated by public operations for drop and form members

Relationships

• M:N relationships

```c++
class People{
  ...
  relationship set <Experiment> makesExperiments
    inverse Experiment::experimenter;
}
class Experiment{
  ...
  relationship set <People> experimenter
    inverse People::makesExperiments;
}
```

- collection types different from set can be used:
  bag, list, array
ODMG Data Model: Operations

- Behaviour: Operations
  - Method specifications (syntax) with in, out, inout parameters
  - Syntax identical to Corba interface definition
  - Exceptions
  - Implementation in binding language

```java
class Experiment{
    attribute string name;
    void writeReport(in: RType typOfRep )raises (noSuchType);
    // specification as comment.
    // implementation: binding language
    ...;
}
```

ODMG Data Model

- Extents
  - Class type may be associated with an extent
  - Extent of class type T: set of all objects of T

```java
class People
    (extent ThePeople
        key name, first_name
    )
    {....
        attribute string name;...
    }
```

- Keys
  - Class type may have a key of one or more of its attributes
  - Values must be unique
ODMG Data Model

• Inheritance
  – Single inheritance supported by classes ("extends")
  – Inheritance of state and behaviour

```plaintext
class People {...
  attribute string name;...
  void writeReport(in: RType typOfRep )raises
    (noSuchType);)
};

class Scientist extends People
{...
  attribute set <subject> fields;
  ...
};
define enum subject (chem,bio,phys,bioInf)
```

• Inheritance and extents
  – Subset relationship of extents to be maintained by system

```plaintext
class People
(extent ThePeople)
{....
  attribute string name;...
  void writeReport(in: RType typOfRep )raises
    (noSuchType);});

class Scientist extends People
(extent Scientists
{...
  attribute set <subject> fields;
  ...
});
```

For each valid DB state $\text{Scientists } \subseteq \text{ThePeople}$
**ODMG Data Model**

- Multiple inheritance by *interfaces* (sic!)

```
Employees -> Tutors

Students -> Tutors

Employees implements interface EmpIF

Students inherits Tutors
```

**3.3 ODMG / Object Query Language (OQL)**

- Language characteristics
  - Query language with SQL flavour
  - Declarative access to objects from application
  
  ```
  define bioScientists as
  select x
  from Scientists x
  where bio in x.fields
  ```

  Defines an extent
  Extent is queried

- No updates: use methods which manipulate state
- Means to traverse the object graph
- OQL may invoke methods programmed in the language of the binding, e.g. Java, C++
OQL typed result sets

- Simple queries and result sets

  ThePeople // collection of all people objects

  select x // has type bag <Employee>
  from Scientists x
  where bio in x.fields

  select distinct x.name // has type set <string>
  from Scientists x
  where set (math,bio) in x.fields

  select distinct struct (a: x.name, b:x.fields)
  from Scientists x // Type: set<struct {a:string, //         b:set<subject>}> //

OQL path expressions

- Navigation through object graph using predefined relations

  "Find name of department which offers the course "Topology"

  Select x.isOfferedBy.worksIn.name
  from course x
  where x.name = "Topology" )

  Only allowed for n:1 / 1:1 relationships
  Example: exactly one prof for course, exactly one dep for prof
**OQL path expressions**

- Navigation through relationship sets

```
Department  hasProf  Professor  teaches  Course
```

"Find professors from all departments with their offered courses, provided the course is an introduction and more than 1 is offered"  

```sql
select *
from departments x,
x.hasProfs as y,  Explicit iterator variables y,z
y.offers as z
where z.name like "%introduction%"
)
```

Result type: bag <struct (x: Department, y: Prof, z: Course)>

---

**OQL methods in queries and subselects**

"Find employees x with higher salary than 10K and for each x those who are paid higher"

```sql
select
  struct (n: x.name,
    hiPaid : (select y from employees
      where
        y.sal.monthlyPayment > x.sal.monthlyPayment ))
from employees x,
where x.sal.monthlyPayment > 10000
)
```

Result type: bag <struct (x: string, y: bag<Employee>)>

x.y and x -> y are equivalent notations
### OQL collection expressions

- **Quantifiers**
  - \( \text{for all } x \in e_1:e_2 \)
  - \( \text{exists } x \in e_1: e_2 \)
  
  are expression, if \( e_1 \) denotes a collection and 
  and \( e_2 \) is an expression of type boolean

  ```
  select x.name
  from Experiments x
  where \text{for all } y \in x.experimenter: math in x.fields
  ```

- **Aggregation**
  - \( \text{max, min, sum, avg} \) may be applied to collection \( e \) of 
  numeric type
  - \( \text{count (select x.Experimenters from Experiments x)} \)
  - \( \text{count} \) can be applied to any collection type object

### OQL Partitioning

- **Partitioning by means of group by**
- **Groups may be referred to by partition variables**
  
  find average salary of all departments with more than 10 emps

  OQL:

  ```
  select deptName,
      \text{avg(select p.sal.monthlyPayment from partition p)}
  from employees e
  group by deptName: e.dep.name
  having count (partition) > 10
  ```

  Type of result of group by operator:

  \[
  \text{type( group by } f_1:e_1, \ldots f_k:e_k) =
  \text{set <struct } (f_1:t_1, \ldots, f_k:t_k, \text{ partition: bag<B>}>)
  \]

  and

  \[
  \text{type( } e_i \text{ ) } = t_i, \text{ type( partition) } = B, \text{ where B is type of from-clause}
  \]
OQL: more language features

- Set expressions
  - union, except, intersect
  - employees except profs

- Indexed expressions:
  - e of type list(t) or array(t), e' of type integer,
  - then e[e'] is of type t
  - and x[i] denotes the i+1 – th element of the indexed collection x

- Joins?
  - Necessary at all?
  - Yes, if not all relationships anticipated in the schema

```sql
select p from Prof p, Student s where p.name = s.name
```

OQL Polymorphism

- Polymorphic type system of ODL
  - Late binding to carry out operations on polymorphic collections
  - Single dispatch

```sql
select struct (n: e.name, e: p.sal) // assumed: operation sal
from People e
```

`sal` operation is dispatched according to dynamic type of an `People` object.
OQL

• Type specifiers
  ... as in Java and other oo languages

  ```
  select  ( (Scientist) e).subject
  from   People e
  where  e.sal.monthlyPayment  >  4000
  // if only scientists earn more than 4000
  ```

• Constructing objects

  ```
  People(name: "Abel", id: 23, sal:
    Salary(1000,0,0,0,0), fields: set(bio,math),
    makesExperiments: set())
  ```

OQL language design

• OQL is a purely functional language

• SQL expressions
  – Ad hoc  - in contrast to relational algebra

• OQL expressions:
  – Typed
  – Freely composed by operators
    - as long as type system is respected –
  – => Orthogonality of the language
**OO database systems**

- **Systems**
  - Poet, Ontos, O2, ObjectStore, ...
  - Do not implement all features of ODMG standard
  - Decreasing importance (?)

- **Significance**
  - Useful for modeling complex domains
  - Object relational systems more important
  - Object relational Mapping architectures use Standard RDBS

---

**3.4 Persistent objects**

- **Design principles**
  - Keep language clean from DB access code
  - "persistence manager" between client code and DB
  - DB specific code generated for use by the persistent mgr

![Diagram of Client space and Persistence layer with DB access code and types allowed: Relational, Object oriented, Object relational]
Persistence Framework

- Mapping between DB data and objects defined in a Meta Data Repository (XML document, database, …)
- Basic functionality: create, read, update, write, transaction support
- Enhanced: fault tolerance, mapping generation, caching etc.

Persistence Framework

- Features
  - Implicit persistence:
    - framework automatically makes business objects persistent
    - no action taken by application program
      - Examples: Enterprise Java Beans, Java Data Objects (container managed)
  - Explicit persistence
    - application program indicates when to save
      - Most common in Persistence frameworks
  - DB access generated
    - dynamically: flexible
    - at compile time: simpler, better performance
    - during system start up
      (example: Hibernate framework)
3.5 OPM

• Object Protocol Model
  – data model for genomic data (1993)
  – subset of oo data models
    • no inheritance, set operations, explicit relationships, ....
  – "protocol classes"
    • intended for modeling the flow of data / control
      in experiments
    • useful for modeling steps and their interrelationships
      of experiments
  – Implemented on top of relational (or oo) DBS

Protocols

• Experiments
  – process chain / graph of activities each having
    • input
    • attributes of the activity (what, when, ... dependent on activity
    • output
  – relational / object oriented modeling?
  – OPM designed to model data and data flow,
    not operations
  – basic innovation: interconnection of activities
Example

Figure 2: Diagrammatic Representation for the LABEL Protocol

cf. Chen, Markowitz: the Object-Protocol Model, TR LBL-32738, Lawrence Berkeley lab

Example: Definition of a Protocol

```
PROTOCOL CLASS LABEL
  EXPANSION: STAIN or ([SOUTHERN_BLOT], HYBRIDIZATION)
  ATTRIBUTE electro_gel: ELECTRO_GEL input
  ATTRIBUTE labeled_separation: STAINED_GEL or PROBED_FILTER output

PROTOCOL CLASS STAIN
  ID: stain_experiment
  ATTRIBUTE stain_experiment: INTEGER single-valued
  ATTRIBUTE electro_gel: ELECTRO_GEL input
  isa LABEL.electro_gel
  ATTRIBUTE stained_gel: STAINED_GEL output
  isa LABEL.labeled_separation
```
Example (cont)

```plaintext
PROTOCOL CLASS SOUTHERN_BLOT
  ID: blot_experiment
  ATTRIBUTE blot_experiment: INTEGER single-valued
  ATTRIBUTE electro_gel: ELECTRO_GEL input
    isa LABEL.electro_gel
  ATTRIBUTE filter: FILTER output

PROTOCOL CLASS HYBRIDIZATION
  ID: hybr_experiment
  ATTRIBUTE hybr_experiment: INTEGER single-valued
  ATTRIBUTE gel_filter: ELECTRO_GEL or FILTER input
    isa LABEL.electro_gel or FILTER from SOUTHERN_BLOT via filter
  ATTRIBUTE probe: PROBE input
  ATTRIBUTE probed_filter: PROBED_FILTER output
    isa LABEL.labeled_separation
```

Summary

- Object models much more flexible than relational
- OO programming languages have (different) underlying models
- OO DB model standardized (Definition ODL, Query OQL)
- Mapping to language model essential
- Significance decreasing with object-relational mapping frameworks
- ... and with XML?