Object Oriented Databases

Vincent P. Wade
Dept. of Computer Science
Trinity College Dublin

Outline

1. Rational for OODBMSs
2. Objectives of OODBMSs
3. Object Models
4. Object Query Languages
5. The Future of OODBMSs
6. Example OODMBS - ObjectStore

Software Crisis ?

- Most s/w is delivered late and over budget
- Impossible to make major changes without a total rewrite
- Structured programming based on top-down approach helps
- CASE and 4GLs helps but ... .
- Sharing of data is a violation of modular programming
- Data should be modularised along with programs (encapsulation)

From Whence OODBMSs ?

- OO technology is not new; Simula, an OO programming language for simulation, was developed in Norway in 1960's
- First "pure" Object Oriented Programming language was Smalltalk.
- Also hybrid OO/conventional programming languages e.g. C++

From Whence OODBMSs (cont)

OODBMS emerged from 2 different fields:

1. Programming language community - concerned with adding DB features to Object Oriented Programming Languages
   = OODBMS purists who wish to develop an Object Oriented DB from scratch e.g. O_2, Object Store
2. DB community wanting to extend DBMS with enhanced semantic data modelling concepts.
   = Extension of Relational Model to provide object management facilities e.g. Postgres

Quick Review of RDBMS and their shortcomings

- RDBMS provide efficient management of large amounts of shared data
- RDB technology meets the need of traditional (business) applications:
  - â simple data types
  - â structured data
  - â efficient data access
  - â data sharing and security
- Not so good at emerging new applications (e.g. CAD/CAM, AI, MM)
  - â richer DB programming languages (types)
  - â complex type structures & long unstructured types
  - â General rules and assertions
  - â support for generalisations and aggregations

Quick Review of RDBMS and their shortcomings (cont)

- Impedance mismatch
  - â RDBMS well suited to *ad hoc* query mode, but not application development
  - â application development requires communication between a relational query language (QL) and programming language (PL)
  - â two types of language do not mix well
- Performance
  - â In some cases RDBMS performance not suitable for certain application types e.g. simulation, CAD

Objectives of OODBMS

- Reduce impedance mismatch by having same (or similar) model for DBMS and Applications
- Provide complex object support & arbitrarily long data
- Support extensible data models
- Ability to represent and manage changes over time (versioning)
- Must also provide the conventional DBMS facilities e.g. concurrency control, query facilities, recovery, security etc...
- :
- :

Solution ?

- Combine OOPL and DBS technologies ????
  - Good one but OO concepts are heavily influenced by programming language requirements.
  - The OOPL object data model requirements are NOT NECESSARILY directly suitable as an OODBMS object data model

=> Some OOPL concepts need to be augmented or mapped into OODBMS concepts.

Current state of OODBMS technology

- Lack of single OODBMS object data model - however there are several proposed 'de-facto' object models e.g. ODMG, SQL3
- Lack of formal foundation
- Strong experimental activity
- Products appearing since 1988/90
- OODBMSs are currently hold niche markets in engineering domains (CAD/CAM), scientific & multi-media

Section 3: Object Models

- **Basic OO concepts:**
  - Object Orientation = object + class + inheritance
- **Object and Object Identifier**
  - Any real world entity is an object, with which is associated system wide object identifier. An object identifier is an internal identifier used by the OODBMS, it is not visible to the application developer or database user.
- **Class**
  - All Objects which share the same set of attributes and methods may be grouped in a class
- **Object Type**
  - An Object Type defines (a) the type of

Object Models (cont.)

- **Encapsulation**
  - model data and operations together (attributes & methods)
- Object has an interface part and an implementation part
  - ***Interface part:***
    - specification of set of operations which can be performed on the object and is the only visible part of the object.
  - ***Implementation part:***
    - has a data part (memory of the object) and an operation part (description in some PL of the implementation of each operation)

Object Models (cont.)

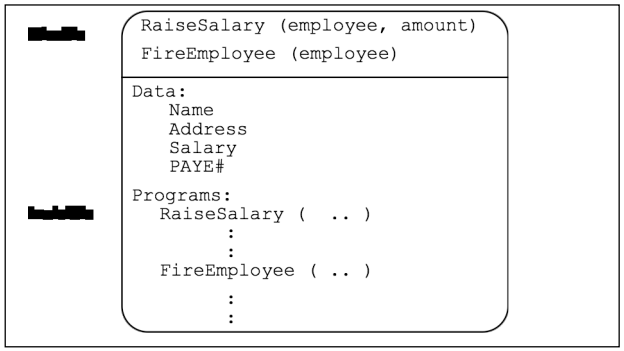
- Encapsulation provides a form of data and operation independence. For example, consider an object 'employee'
- ***In RDBMS:***
  - represented as a tuple
  - queried using relational QL
  - application programmer writes programs (using embedded SQL or 4GL) to update record, raise salary, fire employee etc.
  - programs stored in traditional file system separately from DB
  - **i.e.** there is a sharp distinction between program and data and between QL (for ad hoc queries) and PL (for application programs)

Object Models (Cont.)

- In OODBMSs
  - employee object has a *instance variable(s)* part (similar to tuple of RDB) and an *operation* part which consists of the *raise* and *fire* operations plus some extra operations to construct & destruct the object and to consult the data
  - when storing a set of employees in the DB, data and application programs are stored together

    i.e. there is a single model for data and operations and information can be hidden

Example



Terminology

- Any real-world entity is an **object** e.g. employee. An object can be thought of as an instance of an **object type**.
- All Objects must have a unique, immutable OID
- *name, address, SS#* and *salary* are **instance variables** or **properties** or **attributes** of an object
- *Raise salary* and *fire* are **methods** or **operations** which operate on the values of the attributes of an object

Terminology (cont.)

- The value of an attribute of an object is also an object
- **Typically, this is relaxed in OODBMSs as each object must have an Object Identifier (OID) and if all attribute values were objects this would lead to too many OIDs.**
- *Typically OODBMSs allow the representation of both objects and values*
- An attribute can be single or multi-valued

Encapsulation and Message passing

- messages are sent to an object to operate on the values of the attributes
- all objects which share the same set of attributes and methods may be grouped into a *class or extent*
- *Note:*

  Because a class contains all objects which have the same object attribute and method specifications sometimes the term class is used to identify the "type" of the object .
- *In most OODBMSs, the term class is solely used to determine the set of all objects which are of the same object type!*

Object Identity and Object Structure

- Unlike RDBMSs objects are identified by their system generated OID (In RDBMSs, it is the attributes of an entity that distinguish that entity)
- In other words in OODBMSs, an object has an existence which is independent of its value
  - an object has an existence which is independent of its value
  - two objects can be *identical* (they are the same object)

  or
  - two objects can be equal (they have the same value)

Example: Object Sharing

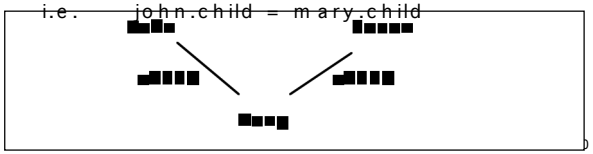
"John" has a child "Mary"

"Susan" has a child "Mary"

with object identity, can model 2 situations:

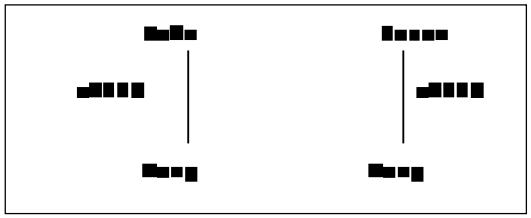
(1) The child of John is *identical* to the child of Susan

i.e. john.child = mary.child



Example (cont)

(2) The child of John is *equal* (but not identical) to the child of Susan



Attributes and methods

- specification of attribute includes its name, semantic integrity constraints, domain
- an attribute may be constructed for atomic values (integers, real numbers, character strings, booleans, dates etc.) or may be constructed from other objects using **type constructors**
- We can think of an object as consisting of a triple (object identifier, constructor, value) where the constructor describes how the complex object is to be put together
- Some example constructors are **atom**, **tuple**, **set**,

Example Type Constructors

- *Set:* A set is a sequence of objects of the same type. No duplicate objects are stored in the set
- *List:* Is similar to a set excepts the OIDs of objects it contains are ordered
- *Bag:* Is similar to a set excepts that it may contain duplicate values to exist

These constructors are also called 'collection types' as they 'collect' objects together

Example

Suppose we are modelling an department which has a department name, department number, manager, locations, employees and projects

- Each manager is associated with a start date.
- A department may have several employees, locations and projects
- Employees have a name, social security number, sex, and belong to a department.

Example (cont.)

Lets take a specific instance of the object department.

- The department name is RESEARCH.
- It is located in HOUSTON, BELLAIRE and SUGARLAND,.
- Its departmental number is 5.
- Its manager's start date was 22-May-78

:

:

Example (cont.)

$o_1 = (i_1, atom, Houston)$

$o_2 = (i_2, atom, Bellaire)$

$o_3 = (i_3, atom, Sugarland)$

$o_4 = (i_4, atom\ 5)$

$o_5 = (i_5, atom, Research)$

$o_6 = (i_6, atom, 22\text{-}May\text{-}78)$

$o_7 = (i_7, set\ \{i_1, i_2, i_3\})$

$o_8 = (i_8, tuple, <DNAME:i_5, DNUMBER:i_4, MGR:i_9, LOCATIONS:i_7, EMPLOYEES:i_{10}, PROJECTS:i_{11}>)$

$o_9 = (i_9, tuple, <MANAGER:i_{12}, MANAGERSTARTDATE:i_6, >)$

$o_{10} = (i_{10}, set, \{i_{12}, i_{13}, i_{14}\})$

$o_{11} = (i_{11}, set, \{i_{15}, i_{16}, i_{17}\})$

OO Schema

- An OO Definition Language (ODL) that incorporates the preceding type constructors can be used to define object types for particular application.
- These object types form the OO database schema

Example OO Schema

```
define type Employee:
    tuple ( name: string, ssn: string, birthdate: date, sex : char, dept:
    department );
define type Date:
    tuple (year : integer, month : integer, day : integer);
define type Department:
    tuple ( dname: string,
            dnumber: integer,
            mgr:  tuple ( manager : Employee, startdate: Date),
            locations : set ( string),
            employees: set (Employee),
            projects:  set (Project) );
```

Note: Attributes that refer to other objects (e.g. dept of employee or projects of department) are actually **references** and hence serve to **represent relationships among objects**

Complex Objects

- By using type constructors and attributes which refer to other objects, we can build Complex Objects.
- E.g. Department is a complex object and as given shown previously, can be represented as a directed graph
- Thus definition of a complex object results in nested structure for the definition of object types
- This graph is known as an *object type hierarchy or class composition hierarchy*

Class Hierarchies

A COMPOSITION (aggregation) hierarchy should not be confused with a CLASS (*inheritance*) hierarchy

- *Note: some literature defines a class to indicate an object type declaration and the operations applicable on that class. Whereas others define a class to be a collection of objects. In this section, we consider a class to just indicate a collection of objects.*
  - **Class hierarchy** captures the generalisation relationship between one class and a set of classes specialised from it (i.e. inheritance)
  - A class is typically defined by its name and by the collection of objects that are included in the class.

A class X is said to be a **subclass** of a class Y, if every object in the subclass X is also in the class Y. In this case Y is

Example

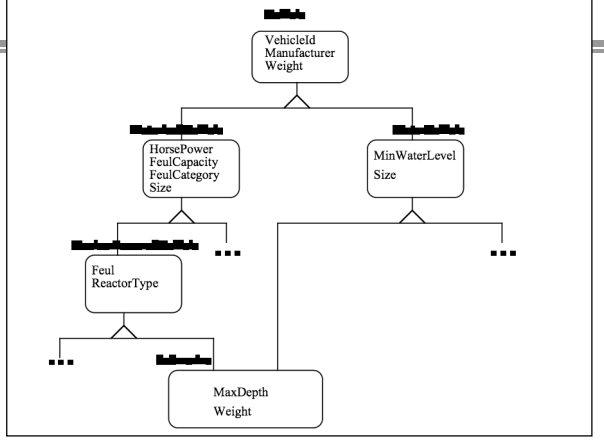
Suppose Y is the class of all mammal, and X is the class of monkeys. If X is a sub class of Y, then every monkey is also a mammal.

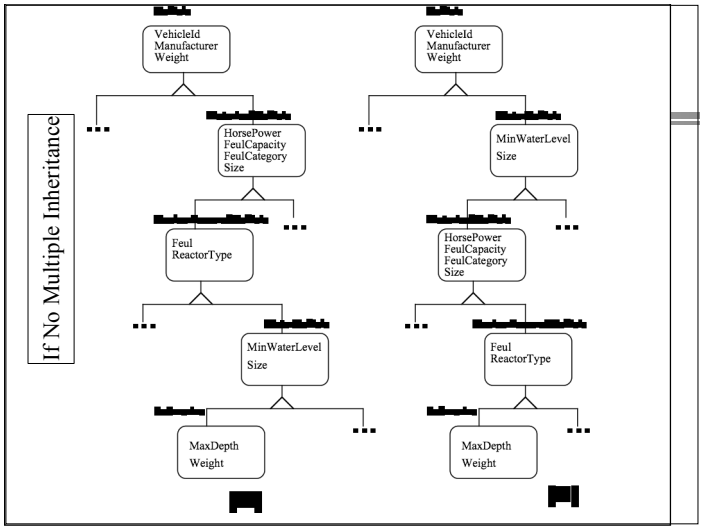
- This subclass/super class relationship is called Specialisation or Generalisation (depending which way you look at it, up or down!)
- Most OODBMSs have a predefined system class (called ROOT or TOP or OBJECT) that contains all the objects in the system
- Classification then proceeds by specialising objects into additional classes that are meaningful to users. => Creation of a Class Hierarchy

Class hierarchy and inheritance

- *single inheritance*:
  - a class may have only one superclass and hence inherits attributes and methods from only one class
- *multiple inheritance*
  - a class inherits attributes and methods from many classes; classes form a rooted directed acyclic graph (DAG) or lattice

Example of Multiple Inheritance

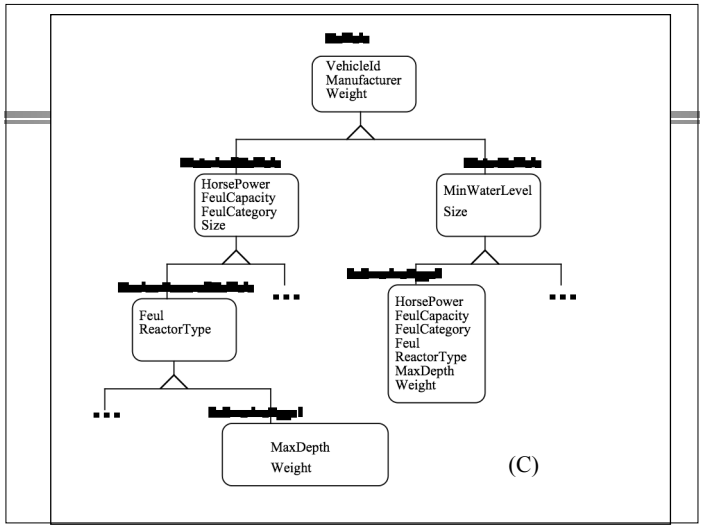




Example where no multiple inheritance

(a) and (b) are 2 alternatives;

- problem is that the extra superclass ends up inheriting attributes and methods from semantically unrelated classes
- e.g class WaterVehicle inherits all its attributes and methods from classes MotorisedVehicle and NuclearPoweredVehicle.
  - However it is clear that class WaterVehicle is not a logical subclass



Example (cont.)

(c) solves this problem, but superclass Submarine must now be modelled in 2 different ways, corresponding to the 2 superclasses it has.

- Each type of Submarine must now replicate attributes and methods that it would inherit from the extra superclasses if multiple inheritance is used

Features of OODBMSs Object Models

- OODBMSs can support persistence of Classes or individual Objects
- OODBMSs allow storage of structured and unstructured complex objects.
- Structured complex objects are (as previously described) where object definitions can

  (i) contain other object definition(s)

  and/or

  (ii) contain reference(s) to object(s) which exist independently.

Features of OODBMSs Object Models (cont)

- Unstructured Objects are objects whose internal structure cannot be interpreted by the database
- In this case it is the applications which interpret the contents of the objects. Examples of these type of objects would include bitmap images or long text strings.
- Such unstructured objects are usually known as Binary Large OBjects or **BLOBs**
- OODBMSs can support versioning of classes level or of individual object(s) (instances)
- Inter Object Relationships 1:1, 1:n, n:m cardinality
- ODBMSs support transactions on objects (see

Section 4: ODBMS Query Languages

- In general ODBMS Query Languages provide a declarative means of searching the database for objects given specific
- Will associatively search DB for objects (remember everything is an object in an OODBMS)
- Usually QL are provided as an extension of the OODBMS Language. In Object Store for example - the ObjectStore DBMS has extended C++ to allow queries to be issued
- Queries (in Object Store) are made on collections of objects (e.g. sets, lists, bags etc.)
- Section 6 (the handout on Object Store) gives

Section 5: OODBMS Achievements and Future!

- â Faster development through reuse; can take longer to program in OO (on line-by-line basis)
- â Higher quality from reuse of tried and tested components and increased modularisation (independence of modules from each other makes testing easier)
- â Easier maintenance through easier detection of faults and repair because OO models the real world more faithfully
- â Reduced cost in programming (through reuse), system design and administration (through

OODBMS Achievements and Future

- u Increased scalability through better modularisation; large scale systems are easier to build and maintain when they are built out of subsystems that can be developed and tested independently
- u Better information structures through complex objects, class hierarchies etc.
- u Increased adaptability; local changes can be made without rebuilding entire system; made possible by message

Potentials Concerns

- u Relatively new technology
- u Need for standards
- u Need for better tools
- u Speed of execution
- u Availability of qualified staff
- u Costs of migration
- u Support for large-scale modularity; modularity in OO is fine grained; no support for combining modules into larger functional units;
- u Composite Objects are useful but they fail to hide internals in the same way as simple objects

OODBMS Standardization

Its a great idea..... but there is so many standards to choose from !!!

- **Current Initiatives for standardized Object Models**
  - SQL 3: Relational Model extended with support for OO
  - ODMG: De Facto standard from a number of OODBMS vendors and OO experts. Affiliated to Object Management Group (OMG) and built upon OMGs Object Model.

It defines Object Model, Object Definition Language, Object Query Language, C++ Language Binding, Smalltalk

Section 6: Example OODBMS - ObjectStore

- Two Papers are MANDATORY for further reading:

  'What Are Next-Generation Database System', R Cattell, published in CACM Oct 1991, Vol 34, No. 10

  Object Store Paper 'The ObjectStre Database System' C Lamb, G Landis, J Orenstein, D Weinreb, published in CACM Oct 1991, Vol 34, No. 10

References

- 'What Are Next-Generation Database System', R Cattell, published in CACM Oct 1991, Vol 34, No. 10
- Object Store Paper 'The ObjectStore Database System' C Lamb, G Landis, J Orenstein, D Weinreb, published in CACM Oct 1991, Vol 34, No. 10
- 'The Object Database Standard: ODMG - 93', R Cattell (Ed.), published by Morgan Kaufmann 1994. ISBN 1-55860-3-2-6
- 'Fundamentals of Database Systems', R. Elmasri, S. Navathe, published by Benjamin/Cummings, ISBN 0-8053-1753-8 (Chapter 22)
- 'Modern Database Systems, the Object Model, Interoperability, and Beyond', W Kim, published by ACM Press ISBN 0-201-59098-0