Why OO approach for DB?

- **Complex Data Handling**
  1) Design software such as CAD, CASE, production planning
  2) Binary files such as image, audio, video

Why OO approach for DB? (Cont.)

- **Consistent Data Model**

Relational: Information regarding entity can be spread across several tables.

OO approach: 1-to-1 relationship between real-life entities and database objects representing them.

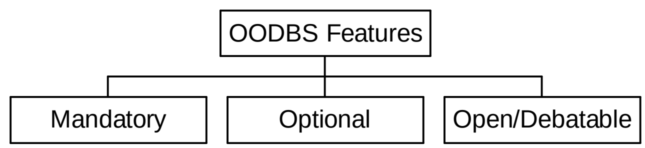
Lack of Specification

Caveat: The paper was written long time ago

- **No single specification, everybody had their own spec. and implementation.**
- **Ill-planned system could emerge and become de-facto standard**

The Novel Idea: A Common Spec

- **OODBS's should have a common set of characteristics**



Mandatory Features

Mandatory OO Features

+

DB Features

=

Mandatory OODBS Features

Mandatory OO Features

1. Complex Objects

Complex Objects can be built from simpler objects using object constructor.

Example: HTML Code

Mandatory OO Features (Cont.)

2. Object Identity (OID)

Each Object has a unique identity

Class Person { string name; int age};

Person assistant, manager;

	Memory Loc.
assistant = ("John", 24)	000111
maneger = ("John", 24)	000222

Two objects are equal but not identical.

Identity maintained by system, not user.

Mandatory OO Features (Cont.)

3. Encapsulation

- **Knowledge about only specification is sufficient for application developer.**
- **Implementation change doesn't affect the applications**
- **Only visible part is interface and allowable functions are only those defined in interface**

Mandatory OO Features (Cont.)

- Classes Vs. Types

1) Type-checking(static vs. dynamic)
2) new
3) extent (optional)

(2) and 3) → Class)

Mandatory OO Features (Cont.)

4. Classes Vs. Types (Cont.)

Run-time	Vs. Compile-time
SmallTalk	C++
Dynamic type-checking	Static type-checking
Class Person {int x; foo();}	
Person john;	
john.bar(); //run-time // error	john.bar(); // compile-time // error

Mandatory OO Features (Cont.)

5. Inheritance

Class Inherit.	Vs. Type Inherit.
SmallTalk	C++
Purpose:	Purpose:
Code-reuse	Interface Compliance Code reuse

Mandatory OO Features (Cont.)

6. Overloading, Late-binding

- Less work at application programmer level
- DBS:
  - ostream display(obj Person);
  - ostream display(obj Image);
  - ostream display(obj Matrix);
- Application Programmer:
  main () { new person; person.sex='M'; display(person);
  new matrix; matrix.size = 4; display(matrix); }

Mandatory OO Features (Cont.)

7. Computational Completeness

- DBS should be able to execute any computable expression fed by Data Manipulation Language
- SQL is computationally incomplete (e.g., doesn't support recursion)

Mandatory OO Features (Cont.)

8. Extensibility

- Predefined types that would come with a DBS should be extensible

Mandatory OO Features (Cont.)

9. Query Facility

- Application programmers or users should be able to query on data
- Methods: Query language, GUI etc
- 1) High Level, 2) Fast, 3) One type works for all applications

DB Features

- 1) Persistence
  -- Data survive the execution time of program creating or manipulating it.
- 2) Storage Management
  -- index mgt., query optimization etc.
  -- required for managing large DB
- 3) Concurrency
- 4) Recovery

Optional Features

- 1) Multiple Inheritance
  -- conflict resolution
- 2) Type Checking and Inferencing
  -- the more the compile-type checking the better
- 3) Distributed
  -- distributed among many computers, need distributed data management

Open Features

Open because: OO-ness is debatable, no general consensus

- 1) Programming Paradigm
  -- Functional/logical/imperative etc.
- 2) Type Formation
  -- Type formers can be added
- 3) Uniformity
  -- Type < - > Object < - > Method

Afterthoughts

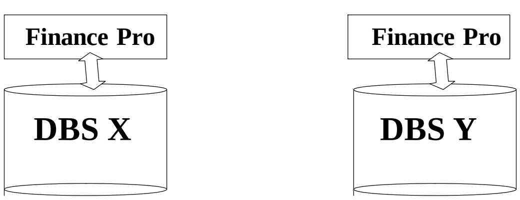
- Most DB Developers are new to OODB Concepts, Relational-to-OO conversion cost, departure from set theory
  -- Can't we extend RDB's to provide the facilities of OODB's?

ODMG-93

- Standard
  -- Portability Vs. Interoperability

ODMG Standard (Cont.)

- Portability
  -- Current OODMS (e.g., O_2, Poet etc.)



ODB Schema

- Relational Schema
  -- Tables/Data structures, Data Types
- ODB Schema
  -- Data structures, Data/Object types, methods(**).

** Programming Lang. needed to write methods

Components of ODMG Model

1) Data Model
2) Language Bindings
3) Query Language

Data Model

- ODB Schema
  -- Three choices: ODL, OMG IDL, OO Language

Data Model(Cont.)

- ODMG(Object Database Mgt. Group) model is an extension of OMG (Object Mgt. Group.
- OMG group deals with object modeling in general, not particular to DB Systems.

Data Model(Cont.)

- OMG General Object Model
  - -- Class
  - -- Instances
  - -- Methods
  - -- Inheritance
  - -- Encapsulation
  - -- Classic Types (e.g., date, time)

Data Model(Cont.)

- DB extension to OMG model

  ⇒ ODMG model

1) Relationship
2) Collection

Data Model(Cont.)

- Collections
  - -- Container for holding instances of a class. Example: Set< Ref<Person> > where Person is a class
  - -- Collection types are template. Example: Set<T>, where T can be any type

Data Model(Cont.)

- Relationship (1-to-1)

```
Class Apartment;
Class Person { String name; Int SSN;
      Ref<Apartment> lives_in inverse is_used_by };
Class Apartment
  {Ref<Person> is_used_by inverse lives_in };
```


Data Model(Cont.)

- Relationships (1-to-many, many-to-many)

```
Class Person {
 Set < Ref<Person> > parents inverse children;
 List < Ref<Person> > children inverse parents; }
```


Object Query Language (OQL)

- Prologue:
  - -- C++ by itself is not enough because
    - Even for a short query, one needs to write, compile, link a C++ file
    - Too much and tedious code

Object Query Language (cont.)

- OQL can help
  - -- Alleviates problems on previous slide
  - -- Extension to host language (e.g., C++). Objects can be manipulated by both.

Object Query Language (cont.)

- OQL Query Example:
  - -- Select m.year
    From Movies m
    Where m.title = “Godfather”
- OQL expression results can be assigned directly to host language variables *C++ variable*

```
Class Movie { …};  Set<Movie> oldmovies;
oldmovies = SELECT DISTINCT m
            From Movies m
            WHERE m.year>1990
```


Object Query Language (cont.)

- Compare the ease of data transfer in OQL with the difficulty in SQL (impedance mismatch)

```
void getStudio()
{
EXEC SQL BEGIN DECLARE SECTION;
    char studioName[50], stuidoAddr[100];
EXEC SQL END DECLARE SECTION;
printf(“Enter studio name”); studioName = getline();
printf(“Enter studio addr”);   studioAddr = getline();
EXEC SQL INSERT INTO Studio(name, address)
            VALUES(:studioName, :studioAddr);
}
```


Object Query Language (cont.)

- OQL’s output type can be complex

```
SELECT DISTINCT struct(star1: s1, star2: s2)
FROM Stars s1, Stars s2
WHERE s1.address = s2.address AND
                         s1.name<s2.name
```

- Recursive OQL Query

```
Set < Ref<Person> > Person::ancestors()
{
Set < Ref<Person> > result;
oql(result, "flatten(select distinct a->ancestors from $1c as a) \
                              union $1c", parents);
return result;
}
```


Afterthoughts

- Schema changes in OODB result in a system-wide recompilation.

  How to solve the problem ?
- Is OQL really impedance mismatch free?

  Syntax, binding stage, programming style are still different.