OBJECT DATABASE MANAGEMENT SYSTEMS: THE ODMG STANDARD

ODBMS: A Short History

- 1984: birth of ODBMS concepts (making Smalltalk a DBMS)
  - D. Maier & G. Copeland
- 1989: definition of ODBMS (Manifesto)
  - M. Atkinson & F. Bancilhon & D. DeWitt & K. Dittrich & D. Maier & S. Zdonik
- Early 90’s: First Products
  - Gemstone, ObjectStore, O2, ObjectivityDB, etc.
- 1991: First ODMG proposal for standardization
- 1993: ODMG’93 Release 1.2
  - R Cattel (JavaSoft), T. Atwood (Object Design), D. Barry (Barry & Associates, Inc.), J. Duhl (Ontos), G. Ferran (O2Tech), M. Loomis (Versant), D. Wade (Objectivity), D. Jordan (Lucent), J. Eastman (Windward Solutions)
- 1995: ODMG Release 2.0
  - R Cattel (JavaSoft), D. Barry (Barry & Associates, Inc.), D. Bartels (POET), M. Berler (AMS), J. Eastman (Windward Solutions), S. Gamerman (O2Tech), D. Jordan (Lucent), A. Springer (Servio), H. Strickland (Versant), D. Wade (Objectivity)
The Object Data Base Management Group

- Founded in 1991 by R. Cattel (Sun) and 5 ODBMS vendors
  - Members: Objectivity, Object Design, Ontos, O2 Tech, Versant
  - Reviewers: DEC, HP, Intelletic, Itasaca, Poet, Servio, Texas
- Allows the portability of applications written for object databases across most of the available ODBMS
- Provides standard interfaces
  - Object Data Model
  - Object Definition Language (ODL)
  - Object Interchange Format (OIF)
  - Object Query Language (OQL)
  - Application Programming Language Bindings: C++, Java, Smalltalk
- Adopted by OMG (IDL, CORBA), studied by INCITS (SQL3)

ODMG Architecture

- Language Interface
  - OQL
  - C++ ODMG
  - Smalltalk ODMG
  - Java ODMG
- Object Database Engine
I) ODMG Data Model

ODMG Principles

- Structural approach:
  - Building complex types (NF2)
  - Distinction between object and value (literals)
  - Class and/or Type

- Behavioral approach:
  - Encapsulation principles (ADTs)
  - Operations (constructor, destructor, read and write)
  - Inheritance hierarchy
  - Overloading and late binding

- Type = specification+implementations
The external specification of a type consists of an implementation-independent, description of its properties + operations + exceptions.

- An interface specification defines only the abstract behavior of an object type.
- A class specification defines the abstract behavior and abstract state of an object type.
- A literal specification defines only the abstract state of a literal type.

The implementation of an object type consists of a representation and a set of methods:

- The representation is a data structure that is derived from the type's abstract state by a language binding (e.g., C++, Java).
- The methods are procedures bodies that are derived from the type's abstract behavior by a language binding (e.g., C++, Java).

### ODMG Types

- **Type**
  - **Literal**: Atomic Lit., Collection Lit., Structured Lit.
  - **Object**: Atomic Obj, Collection Obj, Structured Obj

- **Atomic Lit.**: long, short, ulong, short, double, character, boolean, string, octet
- **Collection Lit.**: set, bag, list, array, dictionary
- **Structured Lit.**: date, time, timestamp, interval, structure
- **Atomic Obj**: Set, Bag, List, Array, Dictionary
- **Collection Obj**: Date, Time, Timestamp, Interval
**ODMG Objects vs. Literals**

- **Object type:**
  - user-defined concepts e.g. Artist, Artifact
  - not printable
  - updateable
  - not built-in
  - objects created by appl.
  - operations defined
  - identity independent of state, sharable (cycles)

- **Literal type:**
  - widely known concepts e.g. integer, date
  - printable
  - not updateable
  - built-in
  - \( \infty \) values pre-exist
  - operations predefined

---

**ODMG Data Model: An Example**

```
creator  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
located in  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
exhibits  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
influences  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
influenced by  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
many-to-many  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
one-to-many  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
one-to-one  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
extends  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
is-A  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

---

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
ODMG Interfaces and Classes

- **Interfaces** define the abstract behavior of application objects
  - are types that cannot be directly instantiated
  - Interfaces may inherit from other interfaces
- **Classes** are a subtype of Interface whose properties define the abstract state of objects in an ODBMS
  - are types that can be directly instantiated
  - may inherit object behavior from interfaces and may extend the object state of other classes
- The ODMG model supports **multiple inheritance** of object behavior and state
  - conflicts (operations with same names but different arguments, properties with the same name) are prohibited (imply a renaming)

---

**ODMG Interfaces and Classes: Examples**

```java
interface Person
{
    attribute string name;
    attribute date born;
    attribute date died;
    short age();
}

class Artist : Person
{
    attribute string name;
    list <Artifact> artifacts;
    relationship list <Artist> influences;
    inverse Artist::influenced;
    attribute string Nationality;
}

class Artifact
{
    extent artifacts key title, created
    {
        attribute string title;
        attribute string technique;
        relationship Artist creator;
        inverse Artist::artifacts;
        attribute date created;
        attribute relationship Museum located_in;
        inverse Artifact::exhibits;
    }
}

class Museum
{
    extent museums key denomination
    {
        attribute string denomination;
        attribute struct {number:int, street:string, city:string, country:string, zip:string} Address;
        relationship set <Artifact> exhibits;
        inverse Museum::located_in;
    }
}
```
**ODMG Object Characteristics**

- A type defines a set of characteristics (properties + behavior) through which the state of its instances can be manipulated.
- Two kinds of properties:
  - **Attributes** of a given type
  - **Relationships** between two types (only for object types)
- The **behavior** of type instances is specified as a set of operations (Methods) signatures:
  - Operations are defined only on a single type
  - Exceptions in the operations of a type can be also provided

**ODMG Relationships**

- Three kinds of relationships: 1-1, 1-N, N-M
  - A relationship is defined by the declaration of a traversal path
  - **Traversal paths** can be declared in pairs one for each direction of the traversal of the binary relationship
  - **Traversal paths** may lead to **ordered** or **unordered** collections of objects as it is indicated by the collection type specification in the relationship declaration
- OBMG compliant DBMSs are responsible for maintaining the referential integrity of relationships
  - The implementation of relationships is encapsulated by public operations that form and drop members from the relationship, plus public operations on the relationship target classes
N-M Relationships

- **n-mN**: A relationship between class a and class b, where a can have multiple instances of b, and b can have multiple instances of a. The relationship is bi-directional.

- **n-mA**: A relationship between class a and class b, similar to n-mN, but with a dotted line indicating a different direction or nature of the relationship.

- **n-mM**: A relationship between class a and class b, again similar to n-mN, with a dotted line indicating a different relationship type.

**ODMG Extents and Keys**

- **An extent is an optional property for a class**
  - The extent contains all the instances of the class and its subclasses.
  - The extent of a subclass is a subset of the extent of its superclasses.
  - Class extents are maintained by the ODBMS.

- **A key can be associated with an extent**: it is a complementary way to identify an object or to retrieve an object.
  - Object values can be identified by associated keys (simple or compound).
  - The scope of uniqueness is the extent of the type (keys only in extents).
  - A key is an integrity constraint to be checked by the system.
ODMG Metadata

- **Descriptive information** about database objects that defines the schema of a database
- **Relationships between metaobjects**
  - guarantee referential integrity of the metaobject graph
  - do not guarantee semantic integrity or completeness
- **Specific operations for schema maintenance** (valid schemas)
- Compared to the OMG IDL repository

**ODMG Metadata Model**

- **Class**
  - key_list
  - extent_name
  - super_class
  - extends
  - has

- **Object**
  - OID
  - names
  - class
  - create
  - delete
  - copy
  - exits
  - same_as

- **Property**
  - **Attribute**
    - attr_name
    - attr_type
    - set_value
    - get_value
  - **Traversal path**
    - path_name
    - to_cardinality
    - to_type
    - traverse
    - creator_iterator

- **Operation**
  - signature
  - invoke
  - return
  - return_abnormally

- **Relationship**
  - add_member
  - remove_member

+ Type, literals, interface, ....
Interfaces for Collections & Iterator

```java
interface Collection : Object {
    exception ElementNotFound ();
    boolean is_empty ();
    boolean contains (in any element);
    void insert_element (in any element);
    void remove_element (in any element)
        raises (ElementNotFound);
    Iterator create_iterator();
}
```

```java
interface List : Collection {
    exception InvalidIndex();
    any retrieve_element_at (in unsigned long index)
        raises (InvalidIndex);
    void remove_element_at (in unsigned long index)
        raises (InvalidIndex);
    List concat (in List alist);
    void append (in List alist);
}
```

```java
interface Iterator : Object {
    exception NoMoreElement ();
    boolean at_end();
    any get_element ();
    raises (NoMoreElement);
    void next_position
        raises (NoMoreElement);
}
```

```java
interface Set : Collection {
    Set create_union (in Set aset);
    Set create_intersection (in Set aset);
    Set create_difference (in Set aset);
    boolean is_subset (in Set aset);
}
```

ODMG Exception Model

- Operations can raise exceptions and exceptions can communicate exception results
- Exception Control:
  - The programmer declares an exception handler within a scope capable of handling exceptions of type t
  - An operation within a contained scope may raise and a exception of type t
  - The exception is "caught" by the most immediately scope that has an exception handler
  - When control reaches the handler, the handle may either decide that it can handle the exception or pass it on (reraise it) to a containing handle
- **NOTE**: ODMG Exceptions are themselves objects and can have an interface that allows them to be related to other exceptions in a generalization/specialization hierarchy
ODMG ODL

- ODL should support the semantics of the ODMG object model
- ODL should be a specification language for interface signatures - not a full programming language
- ODL should be programming language independent
- ODL should be compatible with OMG’s IDL
- ODL should be extensible
- ODL should be practical
  - “supportable by the ODBMS vendors in a relatively short time”

ODMG ODL & Other Standards
OIF: Object Interchange Format

- Specification language used to dump and load the current state of an ODBMS to a file or a set of files.
- Compliant to ODMG/ODL, ~ STEP, INCITS
- State of a database object:
  - object identifier = object tag name
  - type binding = type safe object tags
  - attribute values = tuples, collections
  - links to other objects = references
- Physical clustering
  - example: Monet(Manet) Artist{}

Example: Monet Artist {
  name "Claude Monet",
  style "Impressionism",
  influences{
    Artist Boudin;
    Artist Manet},
  live-time "1840-1926",
  nationality {country: "France",
  city: "Paris"}
}

II) ODMG PERSISTENCE, CONCURRENCY AND TRANSACTION MODELS
ODMG Persistence

- Persistency is orthogonal to types
  - A type may have some instances that are persistent and others that are transient
- Behavior transparency of persistence
  - Both persistent and transient objects can be manipulated using the same operations
- How an object becomes persistent depends on the programming language (orthogonality of persistency and object creation, persistency propagation)
  - In C++, at creation time `a = new(database) Artist`
  - In Java, by attachment to a persistent root

ODMG Locking and Concurrency Control

- Pessimistic concurrency control
  - 2-phase locking
- Locks (~ OMG Concurrency Control Service):
  - Read (shared access to an object)
  - Write (exclusive access to an object)
  - Upgrade (compatible with read but conflict with upgrade & write locks)
- All locks (Read, Write and Upgrade) are held until the transaction is either committed or aborted (compatible with SQL-92 transaction isolation level 3)
  - No dirty reads
  - No unrepeatable reads
  - No phantoms
The ODMG supports both implicit and explicit locking:
- **Implicit** locks can be acquired during the traversal of an object graph (no specific operation is required to obtain a lock).
- **Explicit** locks are acquired by expressly requesting a specific lock on a particular object (lock and try_lock operations).
- **Read** and **Write** locks can be obtained implicitly or explicitly. **Upgrade** locks can only be obtained explicitly.

Programs using persistent objects are organized into transactions:
- Transactions obey the classical properties (**ACID**):
  - **Atomicity** (transactions either finish or have no effect at all)
  - **Consistency** (guarantees consistent transition between database states)
  - **Isolation** (guarantees that concurrent users see an internally consistent database state)
  - **Durability** (the effects of committed transactions are preserved)
  - **Serializable** (concurrent transactions yield results as if they were serial ones)
ODMG Transaction Model

- At most one transaction / thread
- The Transaction interface
  - `begin`, `commit`, `abort`
  - `checkpoint`: locks are not released
  - `join`: associates a transaction object with a thread
  - `leave`: disassociate a transaction object with a thread

```
interface Database {
    void open(in string database_name);
    void close();
    void bind(in any object, in string object_name);
    Object unbind(in string object_name);
    Object lookup(in string object_name);
    Module schema();
};
```
OMG/OM vs. ODMG/OM

- Primary objectives
  - OMG/OM: application portability
    - focus on design portability
  - ODMG/OM: one step further
    - source code portability
- ODL is a superset of OMG/IDL
  - add collections and relationships
- OMG refers to ODL as one protocol for manipulating persistent objects
  - Persistent Object Service Spec.
- OMG refers to OQL as one protocol for querying objects
  - Query Service Spec.

V) REFERENCES

- C. Delobel: “The ODMG Standard”, Course Slides, University of ORSAY