Additional ODL Concepts
OO Model to Relational Model
Object-Relational Design
OO vs OR

Text

• New edition/complete book: Section 4.3-4.5
• Old edition:
  – 2.4.1, 2.4.2
  – 2.5.2, 2.5.5
  – 3.2 (all subsections)
  – 3.4.1
Topics

• More on type:
  – Bag, set, and list
  – Array, dictionary, and structures
• Multiway relationship
• Subclasses and inheritance
• Keys
• Extent

Bag, set, and list

• Set: a set of objects of the same type, no repetition allowed, declared by `Set<T>` where T is a type.
• Bag: set of objects of the same type, repetition allowed, declared by `Bag<T>` where T is a type.
• List: set of ordered objects of the same type, repetition allowed, declared by `List<T>` where T is a type.
Array, dictionary, and structures

- Array: a list whose length is predefined, declared by \texttt{Array<T,i>} - a list of \( i \) elements of type \( T \)
- Dictionary: declared by \texttt{Dictionary<T,S>}, whose value is a set of pair, each pair consists of a value of type \( T \) (key) and a value of type \( S \) (range)
- Structure: \texttt{Struct N \{T_1 F_1, \ldots, T_n F_n\}} (\( T_i \) are types and \( F_i \) are names)

Rules on types of relationships and attributes

- Type of a relationship is either a \textit{class type} or a collection type constructor applied to a \textit{class type}.
- Type of an attribute – can be an arbitrary, possible type.
Multiway relationship

```
class Contract {
    relationship Movie theMovie inverse Movie::contractsFor;
    relationship Star theStar inverse Star::contractsFor;
    ...
}
```

In Movie:
```
relationship Set<Contract> contractsFor inverse Contract::theMovie;
```

Convert to binary relationship and follow the rule!

Subclasses

- declared with ‘extends’
  class Cartoon extends Movie {
    relationship Set<Star> voices;
  }
  class MurderMystery extends Movie {
    attribute string weapon;
  }

Inheritance

- Multiple inheritance: list of classes, separated by `:` after `extends`
  
  class CartoonMurderMystery
    extends MurderMystery : Cartoon

- Conflict might arise – different solutions:
  - disallowed
  - specifying clearly from which property are inherited from which class
  - adding new attributes/relationships to avoid conflicts

Extent

- The set of objects of a class that exists at a given time - difference from the class declaration (similar to instance vs schema)

- Can have different name – declared explicitly with the keyword `extent`

- Example:
  
  class Movie (extent Movies)

  (the name of the extent is **Movies**)
Keys

- Optional
- Declared by key word *key*/keys
- Allow relationships to be part of key
- Example:
  class Movie
    (extent Movies key (title, year)
      (another key here)
      (another key here)
  )

Interface

- Class definition with no associated extent
- Declaration with keyword *interface*
- Enable the existence of several (extent of) classes with the same properties
  interface C { …} ;
  class C1 extends C (extent C11) { extra properties };
  class C2 extends C (extent C21) { extra properties };
  Two extents of C (C11 and C21) can exist two together.
From ODL to Relational Design

- Problems:
  - no key
  - non-atomic values for attributes
  - methods in object definition
- Attributes to attributes
  - atomic: straightforward
  - non-atomic:
    - structure
    - set-value (or collection typed value)
- Relationships
- Methods

<table>
<thead>
<tr>
<th>OLD Attributes to Relational Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic: one-to-one correspondence</td>
</tr>
<tr>
<td>Non-atomic: case-by-case</td>
</tr>
<tr>
<td>- Structure: one attribute for atomic component of the structure, renaming if necessary</td>
</tr>
<tr>
<td>- Set: approach to deal with structure can be used</td>
</tr>
<tr>
<td>- Bag: add a new attribute <code>count</code> to distinguish object</td>
</tr>
<tr>
<td>- List: add a new attribute <code>position</code> to record the order of the object</td>
</tr>
<tr>
<td>- Array: introduce attributes for each array</td>
</tr>
<tr>
<td>- Dictionary: similar to set value</td>
</tr>
</tbody>
</table>
Problem

- Obtained relational schemas might not be in BCNF (others as well)
- To get optimal solution: conversion then normalization

ODL Relationships to Relation

- Similar to conversion of E/R relationships to relations: key of two objects
- OLD Relationships come in pairs: one only
- Combining techniques can be used – applicability depends on the multiplicity of the relationship.
- Example:
  ```java
class Movie (extent Movies key (title,year)){
    ...
    relationship Set<Star> stars inverse Star::starredIn;
    ...
  }
```
  we need only one relation starredIn(title, year, starName) even though ‘starredIn’ will appear as a relationship in the declaration of the class Star.
Object-Relational Model

- Relational model with object-oriented features:
  - structured types for attributes (removing ‘atomic’ requirement)
  - methods (applied on user-defined types)
  - identifiers for tuples (similar to OID)
  - references

Nested Relations

- Attributes need not be atomic:

  Stars(name, address(street, city), birthdate, movies(title, year))

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>birthdate</th>
<th>movie</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fisher</td>
<td>Mapple</td>
<td>9/9/99</td>
<td>Star War 1977</td>
</tr>
<tr>
<td></td>
<td>Locust</td>
<td></td>
<td>Empire 1980</td>
</tr>
<tr>
<td></td>
<td>street</td>
<td>city</td>
<td></td>
</tr>
<tr>
<td></td>
<td>street</td>
<td>city</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oak</td>
<td>B’wood</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1980</td>
<td>1977</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1983</td>
<td>1977</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>birthdate</th>
<th>movie</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamill</td>
<td>street</td>
<td>8/8/88</td>
<td>Return 1983</td>
</tr>
<tr>
<td></td>
<td>city</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>street</td>
<td>city</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mapple</td>
<td>H’wood</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1980</td>
<td>1977</td>
<td></td>
</tr>
</tbody>
</table>
References

- Specify by \{^*R\} where R is the referred relation
  \[\text{Stars(name, address(street,city), birthdate, movies(^*Movies))}\]
- This implies that the attribute *movie* of the Stars relation is a set of references to tuples in the Movies relation

### The Stars example with Reference

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>birthdate</th>
<th>movie</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fisher</td>
<td>street</td>
<td>city</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mapple</td>
<td>H'wood</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Locust</td>
<td>Malibu</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9/9/99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamill</td>
<td>street</td>
<td>city</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oak</td>
<td>B'wood</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8/8/88</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star War</td>
<td>1977</td>
</tr>
<tr>
<td>Return</td>
<td>1983</td>
</tr>
<tr>
<td>Empire</td>
<td>1980</td>
</tr>
</tbody>
</table>
Object-oriented vs Object-relational

- Similar features:
  - Objects in OO model could be represented as tuples in OR model (due to the introduction of set and references)
  - Extents and relations: use interface to overcome the restriction of 'one extent with one class'
  - Methods: implemented in OR DBMS
  - Type systems: almost the same
  - References and OID: references are not necessary hidden from users, OID does
  - Backward compatibility: OR does but not OO – possible reason for popularity of OR vs OO

- Conversion from OO to OR:
  - simpler than OO to Relational Model
  - mostly straightforward (one-to-one correspondence) – otherwise, use the techniques of OO to Relational