Data Model Issues for Object-Oriented Applications

Slide 1

Liza Blaney

September 30, 1996

Orion

Slide 2

- an Object-Oriented Database System
- designed from the Database point-of-view
- intended for data-intensive applications
  - AI
  - CAD/CAM
  - multimedia document office systems (OIS)
- everything is an *object*

Objects

Slide 3

- an *object* consists of its *state* and its *behavior*
  - State:
    * an *object* has private memory which holds its state
      · the state of a *primitive object* consists of its value, which is also an *object*
      · the state of a *complex object* consists of the values of the object's *instance variables*
    * the concept of an *instance variable* is recursive; an *instance variable* is an *object* and has state defined in its private memory by either a collection of *instance variables* or a value

Objects - continued

Slide 4

- Behavior:
  - the *behavior* of an object is captured in the *messages* to which the object responds
  - the *message* completely defines the semantics of the *object*; an *object* can only be created, interrogated, or changed through a *message*
  - each *message* has a corresponding *method*, code that manipulates or returns the state of the object
  - an *object* reacts to a *message* by executing the corresponding *method* and returning an *object* in response

Objects are Grouped into Classes

Slide 5

- *objects* which are described by the same *instance variables* and *methods* can be grouped together into a *class*
- *objects* which belong to the same *class* are called *instances* of the class
- an *object* can belong to only one *class*

Classes

Slide 6

- a *class* defines the *instance variables* and *methods* of its *instances*
- a *class* inherits the properties (*instance variables* and *methods*) of its immediate *superclass*
- by induction, a *class* inherits the properties of every *class* in its *superclass* chain
- because multiple class inheritance is allowed, the *class hierarchy* is generalized to a *class lattice* (a *D*irected *A*cyclic *G*raph)
- however, an *object* may still be the member of only one *class*

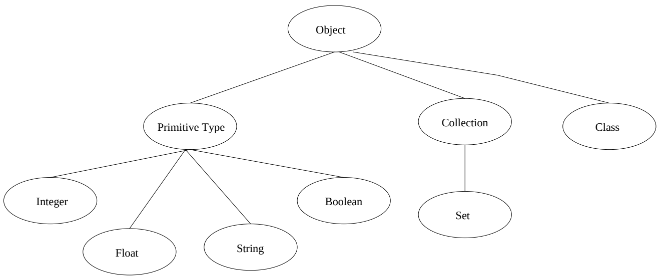
More on Classes

Slide 7

- the root of the DAG is the class *Object*
- a *class* is instance of an *object* of class *Class*
- to create a *class*, a *message* is sent to the *object* of class *Class* requesting the creation of a new class

Directed Acyclic Graph

Slide 8



Typing

Slide 9

- typing is accomplished by binding the domain of an *instance variable* to a specific *class*, e.g.
  - a *class* representing a primitive object, such as "integer"; this allows typing as in a conventional programming language
  - a *class* represented by instance variables and methods, such as "manufacturer"; this allows constraint checking
- an instance variable in a class does not have to be bound to a class; different instances of the same class may use that instance variable to reference objects in different classes

The Lattice

Slide 10

- the *class lattice* defines what in a relational database are the tuples and the relations between them, i.e. the *Schema*
- Orion permits changes to be made to the class lattice without requiring exclusive access to the database
- the *nodes* of the lattice are the *classes*
- the *edges* of the lattice as the relationships between the *classes*
- the *contents* of a node are the *instance variables* and *methods* which define the class

Schema Evolution

Slide 11

- Taxonomy - what can be changed in the schema
  - Contents of Node (i.e. *Class Definition*)
    - * Instance Variables
    - * Methods
  - Edge (i.e. *Inheritance*)
  - Node (i.e. *Class*)
- Invariants - the definition of a consistent state of the schema
- Rules - governing the schema change semantics
- Semantics

Taxonomy: Contents of Node

Slide 12

- Instance Variables
  - Add/Drop instance variables
  - Change: Name, Domain, Inheritance, Default/Shared Value
- Methods
  - Add/Drop methods
  - Change: Name, Code, Inheritance

Taxonomy of Nodes and Edges

Slide 13

- Edge
  - Add/Drop edge between superclass and class
  - Change the inheritance order
- Node
  - Add/Drop a Class
  - Change the Name of a Class

Invariants

Slide 14

- the Class Lattice is a rooted, connected directed acyclic graph
- all names in a class must be distinct
- all instance variables and methods have a distinct origin
- a class inherits all instance variables and methods from each superclass, except for identically named objects
- an inherited variable must have the same domain or a sub-domain of the variable in the superclass

Rules

Slide 15

- Default Conflict Resolution
- Property Propagation
- DAG Manipulation

Default Conflict Resolution

Slide 16

- Distinct Origin
  - name conflicts between a class and a superclass are resolved by giving precedence to the definition within the class
  - name conflicts between super-classes are resolved by giving precedence to the definition within the first superclass (node with the lowest labeled edge)
  - the class may specify which object of multiply defined objects to inherit
- Same Origin
  - if one domain is a subset of the other, the more restricted domain takes precedence

Slide 17

- if the domains are the same or disjunct, the first superclass takes precedence

Property Propagation

Slide 18

- when properties of a class are changes, the changes are propagated to all subclasses that had inherited them (i.e. properties not redefined)
- a name change or name addition is propagated only to those subclasses which have not already defined the name (Corollary: if a subclass does not inherit the change/addition, it does not propagate it.)

DAG Manipulation

Slide 19

- Edge Addition - between A and B makes A the last superclass of B
- Edge Removal - between A and B when A is B's only superclass makes B an immediate subclass of all A's superclasses
- Node Addition - the root *Object* is the default superclass, if none are specified
- Node Removal - to delete A: delete edges from A, delete edges into A, then delete A

Semantics of Schema Evolution

Slide 20

- define a new class
- add a new instance variable to a class
- drop a class
- drop an instance variable from a class
- change the domain of an instance variable

Composite Objects

Slide 21

- the authors' claim: the object-oriented model can represent the relationship "is a" in inheritance, not "is part of"
- a *composite object* belongs to a *composite object hierarchy* of exclusive component objects
- composite objects add the concept of dependent objects: an object whose existence depends on the existence of other objects and which belongs to exactly one other object (Warier-Orr analysis)
- the instance objects that constitute a composite object belong to a hierarchy of classes called a composite object schema
- a composite object is also a unit of clustering on storage

Versions

Slide 22

- a method of allowing multiple users to work on an object, making modifications over a period of time, before committing the modifications to the shared database (e.g. RCS)
- a *transient version* is derived from a *working version* belongs to the user who derived it
- a *working version* cannot be updated, except by being replaced by a transient version