Database Versioning

David Landén
Per-Magnus Olsson

Overview

- Motivation
- Common operations
- Example: Gemstone
- Example: O_2
- Exercises

Motivation

- Two major reasons for versioning:
  - Data change and desire to access historical values.
  - Schema change.

General Considerations

- When to convert?
  - Immediate – change everything now.
  - Deferred – change as object is used
- References to deleted objects?

Common Operations

- Add/remove/modify/rename attribute
- Add/remove/modify/rename class
- Make class S a superclass of C
- Remove inheritance between S and C

Example: GemStone

- Object-oriented database management system in mid 80's,
- GemStone's successors are used today in finance and many other areas.
- Influenced by Smalltalk
- Influenced J2EE

Philososphy

- Allow modifications that are
  - Useful
  - Well understood
  - Have reasonable implementation
- → Not all operations are supported

Authorization and Concurrency

- All objects belong to a segment
- Authorization per segment, not per object
- Access to object does not imply access to instance variables
- Access conflicts checked at commit time
- Read-only transactions always succeeds
- Changes visible to other sessions after commit
- Changes made by others visible after current transaction ends

Variables and Constraints

- Indexable – directly accessible through index (c.f. id number)
- Named – accessible through name.
- Anonymous – not accessible through name or index.
- Named or anonymous are constrained, the value of the variable is constrained to be a kind of a given class.

Class Level Operations

- Removing a class unless
  - instances exists
- Adding a class
  - Leaf nodes without restrictions
  - Interior nodes requires specification of superclass and subclass
  - New instance variables must be added separately

Class Level Operations, cont.

- Making a class indexable allowed unless
  - subclass already indexable and indexing the superclass makes the variable constraints inconsistent.
- Making a class not indexable
  - allowed for all instances
  - not propagated to subclasses
  - Exception: subclass must be indexable of superclass is indexable

Variable Level Operations

- Adding named instance variable allowed unless
  - name clash with existing instance variable in class
  - variable exists in subclass
  - variable propagated to subclasses
- Removing named instance variable allowed unless
  - variable exists in superclass
  - not propagated to subclasses

Variable Level Operations, cont

- Renaming of variables allowed unless
  - variable inherited
  - name clash occurs with new name

Variable Constraints

- Specialization
  - New constraint is subclass of old constraint
- Generalization:
  - New constraint is superclass of old constraint
- Specialization allowed unless
- Generalization allowed unless

O_2

- O_2 has Objects and types:
  - A type is defined recursively from atomic types.
  - Class methods coded in O_2C or C++.
  - Attaching an object or type to name in a schema make it persistent.
  - Encapsulation at: class, application and schema level.

O_2

- Schema modifications:
  - Adding/deleting attributes to a class.
  - Redifining the structure of a class.
  - 10 different schema manipulation primitives:
    - creation of a new class
    - creation of new attribute,
    - deletion of attrbitues
    - …

O_2

- Database updates:
  - Conversion functions:
    - Default
    - User defined.
  - Supports both deferred and immediate conversion.

O_2

- Default conversion function example:

```
create schema Car showroom;
class Vendor type tuple ( name: string,
                          address: tuple ( city : string,
                                           street : string,
                                           number : real),
                          sold_cars: list( Car ))
end;

class Car type tuple ( name: string,
                       price : real,
                       horse_power : integer )
end;
```

O_2

- User-defined conversion function example:

```
begin modification in class Car;
delete attribute horse_power;
create attribute kW : integer;
conversion functions;
conversion function  mod_kW (old : tuple(name:string, price:real,
                                         horse_power:real)) in class Car
{
        self->kW = round( old.horse_power / 1.36 );
};
end modification;
```


O_2

- Conversion function types:
  - Simple:
    - Uses only object local information.
  - Complex:
    - Uses other objects in the conversion.

O_2

- Object Migration functions:
  - Single object change:
    - Uses a migrate() method.
    - Only downwards in subclasses.
  - Class change:
    - Migration functions:

O_2

- Implementation:
  - Based on deferred database update.
  - Complex conversion functions complicates the updates.
  - Solution is to save all versions of objects, (screend).
  - An object has its current value and screened value.
  - Conversion and migrate functions can access the screened value.

O_2

- The O_2 schema manager keep track on the evolution of schemas.
  - updates the schema state for each modification.
  - Each class in a schema is represented with a class descriptor, a meta class in O_2.

O_2 – Meta Objects

```
class Meta_class type tuple (
    ...
    type : Meta_type,
    properties : list (Meta_property),
    ...
    cur_tid : integer,
    history : list (Meta_history_entry),
    ...
end;
```

O_2 – Meta History

```
class Meta_history_entry type tuple (
      tid : integer,
      type : Meta_type,
      ex_type : Meta_type,
      struct : list(Meta_property_entry),
      cf : Meta_conversion,
      mf : Meta_migration)
end;
```


O_2– Meta Property

```
class Meta_property_entry type tuple (
      pid : integer,
      sch_state : integer,
      offset : integer,
      type : Meta_type,
      status : {existing, screened})
end;
```

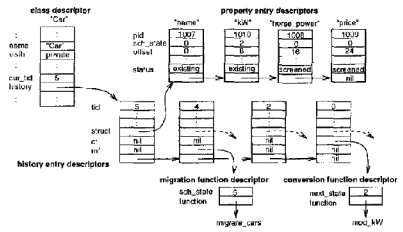

O_2– Meta Conversion

```
class Meta_conversion type tuple (
      next_state : integer,
      function : Meta_binary)
end;
```


O_2 – Meta Migration

```
class Meta_migration type tuple (
      sch_state : integer,
      function : Meta_binary)
end;
```


O_2 – Example



Deferred update Algorithm in O_2

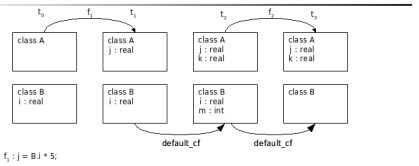
- When accessing an object o:
  - If the object conforms to its last class definition in the schema.
    - Then the object can be used.
    - If not, the corresponding history entry descriptor is found for the object.

O_2

- The history entry descriptor is examined:
  - If there is a migration function.
    - If o is not already migrated use the migration function.
  - If there is a conversion function
    - If there is a conversion function use it.
  - If there is no migration or conversion function.
    - Use default conversion.

O_2

- Problem with complex conversion:
  - When updating an object to time t_n:
    - And accessing other objects in a conversion function could lead to a problems.
    - If the conversion functions was specified in t_m, it could only access other objects of t_m.
    - So other objects must be updated to t_m, when access by this conversion function at tn. ($t_m < t_n$)



f_1 : j = B.i * 5;

f_2 : k = j + B.m;

Questions:
1) An object (instanceof A) conforming to t_2, describe how it is updated at t_3 using the deferred algorithm.
2) An object (instanceof A) conforming to t_2, describe how it is updated at t_3 using the deferred algorithm.