Preserving Semantics when Transforming Conceptual Spatio-Temporal Schemas

Esteban ZIMÁNYI, Mohammed MINOUT
Department of Computer & Network Engineering
Université Libre de Bruxelles
{ezimanyi,mminout}@ulb.ac.be
http://cs.ulb.ac.be

Semantic-based Geographical Information Systems (SeBGIS'05)
November 3, 2005

Contents

- Information Systems Design
- The MADS model
- Translating conceptual schemas
  - Logical schemas
  - Physical schemas
- Preserving semantics when translating schemas
  - Logical constraints
  - Physical constraints
- Examples of spatial and temporal constraints
- Conclusions & future works

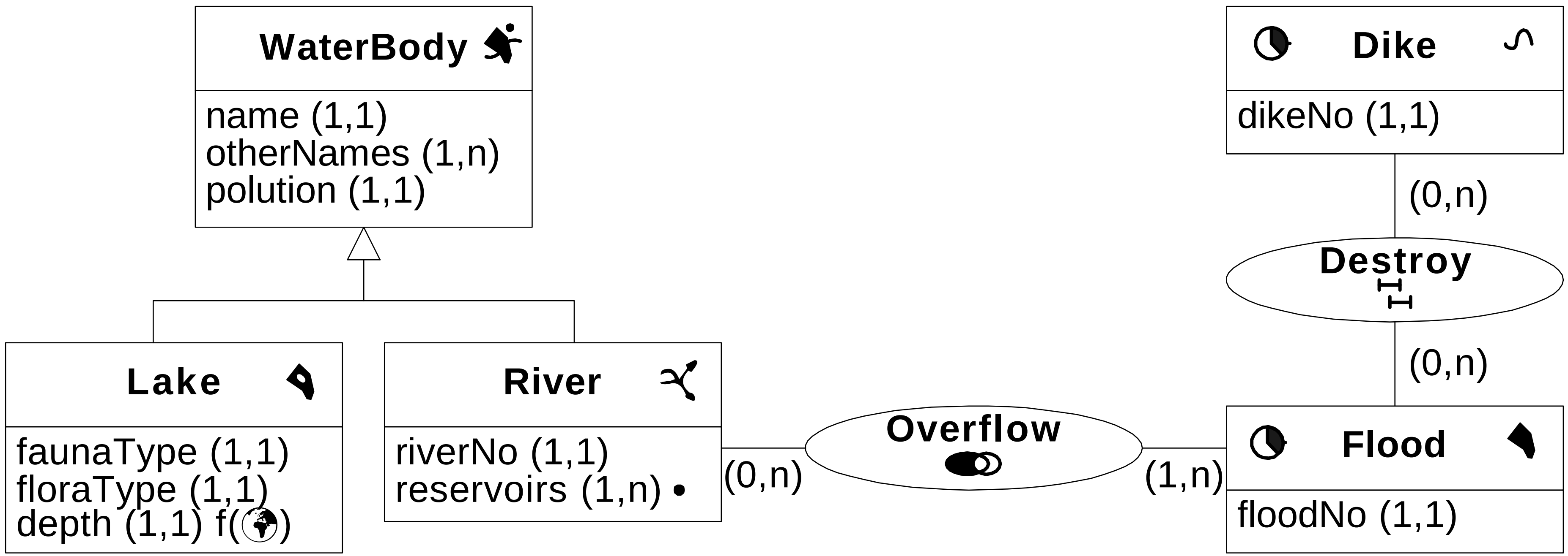
(Geographical) Information Systems Design

- Realized on a three-step approach
  - **Conceptual schema**: Captures application requirements without taking into account implementation considerations
  - **Logical schema**: Targets a family of implementation platforms, e.g., relational, object-relational
  - **Physical schema**: Takes into account particularities of a specific operational platform, e.g., Oracle
- Typically, (semi-)automatic transformation of these levels using a CASE tool
  - **Basic** transformation rules are **simple**
  - **Additional information** must be input at logical and physical levels
  - **Optimization** issues are important and require **human expertise**

The MADS Model

- Conceptual spatio-temporal model with 4 orthogonal modeling dimensions
- **Structural**: novel approach with semantically-rich relationships and multi-instantiation capabilities
- **Spatial** and **Temporal**:
  - based on rich **hierarchies** of data types
  - **orthogonality** for associating spatial/temporal features to types/attributes
  - both an **object-based** and a **continuous views** of space/time
  - **constrained** relationship types: topological, synchronization
- **Multi-representation**: supporting multiple alternative viewpoints on the same information
- Conceptual framework for both **data definition** and **data manipulation**
- C. Parent, S. Spaccapietra, E. Zimányi, *Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS approach*, Springer, 2005, 500p., to appear

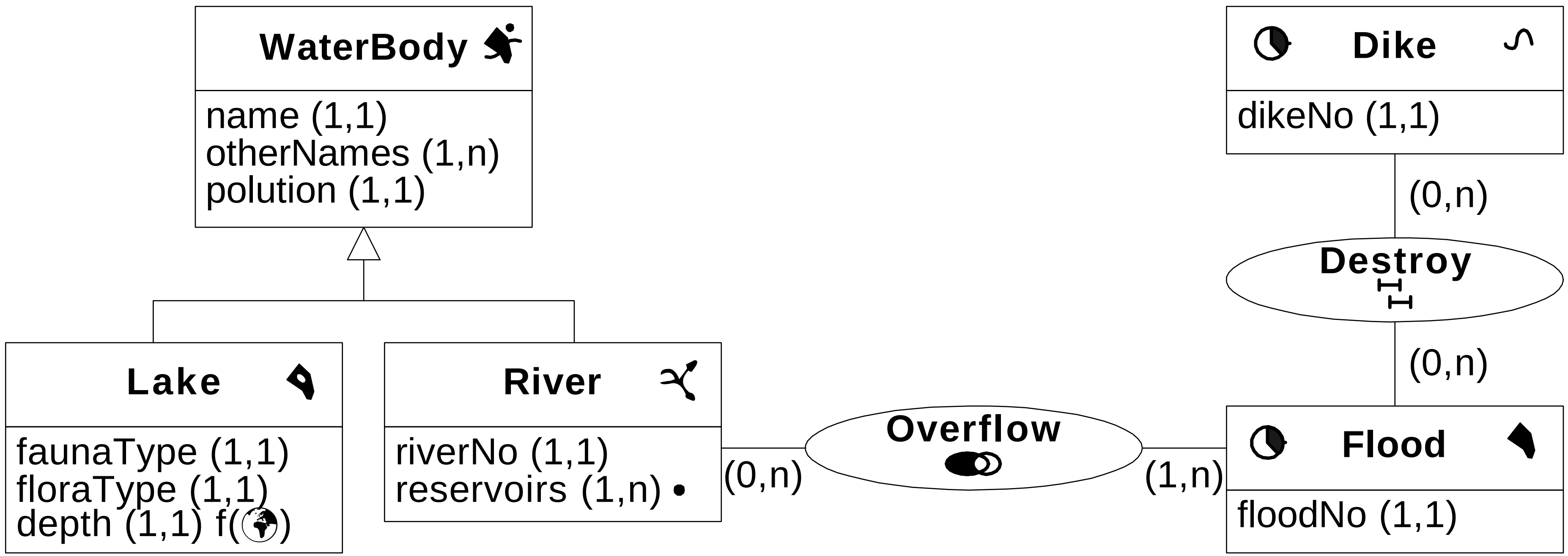
The MADS Model: Example Conceptual Schema



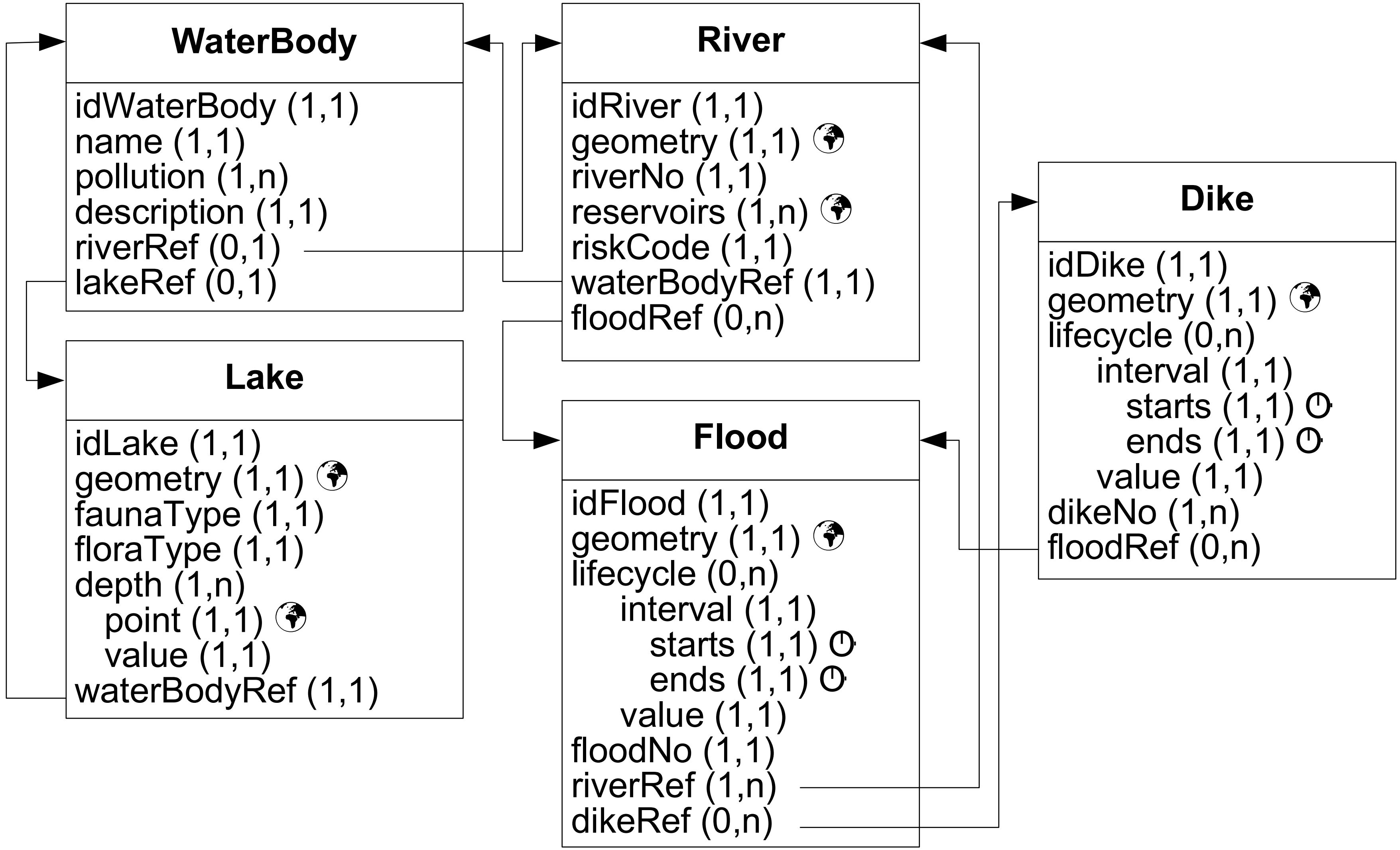
Translating MADS Schemas

- **Transformational approach** replacing rich MADS concept with a set of constructs available in the target implementation platform
  - Logical models: Relational, Object-Relational, ...
  - Spatial models: Oracle, MapInfo, ArcInfo, ...
- **Relationship types**: several rules depending on their types and the cardinality of their roles
- **Spatial O/R types**:
  - materializing predefined `geometry` attribute
  - specialized spatial types $\Rightarrow$ generic one (e.g., for Oracle)
- **Varying attributes**: Complex multivalued attribute encoding its defining function
  - spatial, temporal, and/or perception extent
  - value
- **Temporal O/R types**: Complex attribute encoding the lifecycle, including time-varying status: `scheduled`, `active`, `suspended`, `disabled`

The MADS Model: Example Conceptual Schema



Translating MADS Schemas: O-R Logical Schema



Translating MADS Schemas: Oracle Physical Schema

- Rewriting the logical schema generated by the translator

  ⇒ schema expressed in the language of the target platform

```
create or replace type DInterval as object (
    starts date, ends date);
create or replace type DLifecycleValue as object (
    interval DInterval, status varchar2(10));
create or replace type DLifecycle as table of DLifecycleValue;
create or replace type DRiverSetRef as table of ref DRiver;
create or replace type DDikeSetRef as table of ref DDike;
create or replace type DFlood as object (idFlood Did,
    geometry mdsys.sdo_geometry, lifecycle DLifecycle,
    floodNo integer, riverRef DRiverSetRef, dikeRef DDikeSetRef);
create table Flood of DFlood
    nested table lifecycle store as FloodLifecycleNT
    nested table riverRef store as FloodRiverRefNT
    nested table dikeRef store as FloodDikeRefNT;
[...]
```

Preserving Semantics when Transforming Schemas

- At each step of the transformation some **semantics is lost**
  - ⇒ Data **invalid** in the original conceptual schema is **accepted** by the corresponding physical schema
- **Reason**: Limited expressive power of logical and physical models
- **Integrity constraints** are needed for ensuring the **semantic equivalence** between the conceptual and physical schemas
- Such constraints must be **implemented into the DBMS/GIS**
  - **Encoded once** and for all in the database, instead of being encoded in each application accessing it
  - **Available to all applications** accessing the database, thus enforcing data quality
  - **Encapsulated with the data**, facilitating the overall application lifecycle

Support of Integrity Constraints in SQL:2003

- Choices for implementing constraints
  (1) **Declarative** (built-in) constraints
  (2) **Triggers** which fire upon predefined updates of particular tables
  (3) **Stored procedures** activated by predefined transaction events
  (4) Directly **embedded in the code** of applications
- SQL:2003 provides a few types of **declarative** integrity constraints
  - `NOT NULL`, `DEFAULT`, `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY (REFERENCES)`
  - `CHECK`: defines a general IC that must hold for each row of a table
  - `DOMAIN`: creates a (restricted) column domain
  - `ASSERTION`: defines a named general IC that may refer to more than one table

Support of Integrity Constraints in DBMSs

- DBMSs having an `ASSERTION` statement **do not encourage its use**
- Most DBMSs only support domain, uniqueness, and foreign key constraints
- **Expressive power** of these constructs is quite **limited**, e.g.
  - foreign key constraint: referenced columns must satisfy uniqueness condition
  - domain constraints: tied to single columns only
  - uniqueness contraints: apply only within a single table
- DBMSs recommend to implement user-defined ICs **non-declaratively** (by triggers or stored procedures) for **efficiency reasons**
- ICs typically involve several tables, potentially huge joins, full table scans, nested sub-queries, nested negation, ... $\Rightarrow$ evaluation becomes prohibitively expensive
- Typical OLTP applications and time-critical data warehousing processes **cannot afford** integrity checking

Preserving Semantic Equivalence: Methodology

- Integrity constraints expressed at
  - **Logical level:** first-order formulas that use the methods provided by spatial/temporal data types
  - **Physical level:** declarative constraints or triggers depending on target platform
- **Automatic process** complementing traditional transformational approach
- **Conceptual** ⇒ **Logical**: Each transformation rule associated with a set of logical constraints ensuring semantic equivalence
  - **Result**: Repertoire of logical **constraints patterns**
- **Logical** ⇒ **Physical**: Analysis of implementation possibilites of each constraint pattern of the repertoire

Temporal Constraints: Lifecyle (1)

- Translation of lifecycle requires a set of temporal constraints
- Basic declarative integrity constraints in Oracle

```
alter table FloodLifecycleNT add constraint
    uniqueStarts unique (interval.starts);
alter table FloodLifecycleNT add constraint
    uniqueEnds unique (interval.ends);
alter table FloodLifecycleNT add constraint
    validInterval check (interval.starts < interval.ends);
alter table FloodLifecycleNT add constraint
    validStatus check (status in
      ('scheduled', 'active', 'suspended', 'disabled'));
```

Temporal Constraints: Lifecyle (2)

- The intervals of the lifecycle must be disjoint

$$\forall f \in \text{Flood}, \forall l_1 \in f.\text{lifecycle}, \forall l_2 \in f.\text{lifecycle} \; (\\ l_1.\text{interval.starts} < l_2.\text{interval.ends} \wedge l_2.\text{interval.starts} < l_1.\text{interval.ends} \Rightarrow \\ l_1.\text{interval.starts} = l_2.\text{interval.starts} \wedge l_1.\text{interval.ends} = l_2.\text{interval.ends} \;)$$

- Physical level: triggers in Oracle

```
create or replace trigger FloodLifecycleOverlappingIntervals
before insert on Flood for each row
declare rowcnt number;
begin
  select count(*) into rowcnt
    from table(:new.lifecycle) l1, table(:new.lifecycle) l2
    where l1.interval.starts < l2.interval.ends
    and l2.interval.starts < l1.interval.ends
    and l1.interval.starts <> l2.interval.starts
  if rowcnt <> 0 then
    raise_application_error(-20300,'Overlapping intervals')
  end if;
end
```

Temporal Constraints: Synchronization Relationships

- Synchronization constraints lost in translation

  ⇒ Only underlying binary relationship represented in the schema

- A set of triggers generated automatically for preserving such semantics

```
create or replace trigger FloodDestroySynchronization
before insert on Flood for each row
declare rowcnt number;
begin
  select count(*) into rowcnt
    from table(:new.lifecycle) l1, table(:new.dikeRef) d,
    where not exists (
      select * from table(d.column_value.lifecycle) l2,
       table(d.column_value.floodRef) f
      where f.column_value.idFlood=:new.idFlood
      and l1.interval.starts < l2.interval.ends
      and l2.interval.starts < l1.interval.ends
      and l1.status='active' and l2.status='active' )
  if count <> 0 then
    raise_application_error(-20302,'Violation of synchronization')
  end if;
end;
```

Spatial Constraints: Spatial Types

- If only a generic spatial type (Oracle) ⇒ values of spatial attributes must be of the type in the conceptual schema

- Geometries of rivers are of type multiline or multicurve

```
alter table River
  add constraint validGeometryType check (geometry.get_gtype() = 6);
```

  (not valid in Oracle 10g ⇒ a trigger instead)

- Each value of the attribute `reservoirs` of `River` is of spatial type point

```
create or replace trigger RiverReservoirsPointType
before insert on River for each row
declare rowcnt number;
begin
  select count(*) into rowcnt
    from table(:new.reservoirs) r where r.get_gtype() != 2 )
  if rowcnt <> 0 then
    raise_application_error(-20401,
     'Reservoirs must be of spatial type point')
  end if;
end;
```

Spatial Constraints: Topological (1)

- Topological constraints may relate a spatial attribute with geometry of its type
- The spatiality of `reservoirs` is inside the spatiality of `River`

  $\forall r_1 \in \text{River}, \forall r_2 \in r_1.\text{reservoirs}\ (\ r_1.\text{geometry.within}(r_2.\text{geometry})\ )$

- Trigger at the physical level

```
create or replace trigger RiverReservoirsInside
before insert on River for each row
declare rowcnt number;
begin
  select count(*) into rowcnt
    from table(:new.reservoirs) r
    where sdo_inside(r,:new.geometry)='FALSE' )
  if rowcnt <> 0 then
    raise_application_error(-20402,
       'Reservoirs must be located inside its river')
  end if;
end;
```

Spatial Constraints: Topological (2)

- Topological constraints for relationships are lost in the translation
- `Overflow` is a topological relationship of type `intersect` ⇒ an instance of `River` may be linked to an instance of `Flood` only if their geometries intersect
- Trigger at the physical level

```
create or replace trigger FloodOverflowTopological
after insert on Flood for each row
declare rowcnt number;
begin
  select count(*) into rowcnt
  from table(:new.riverRef) r,
  where not exists ( select * from table(r.column_value.floodRef) f
    where f.column_value.idFlood=:new.idFlood
    and sdo_overlaps(:new.geometry,r.column_value.geometry)='TRUE' )
  if rowcnt <> 0 then
    raise_application_error(-20404,
      'Violation of Overflow topological relationship')
  end if;
end;
```

Space-Varying Attributes

- Many constraints apply to varying attributes
  - depend on the type of the underlying function: **discrete**, **stepwise**, **continuous**
- Attribute `depth` in `Lake`: every value of attribute `point`

  (1) is of spatial type point

  (2) is located inside the geometry of the lake
- Another constraint: the points are at least 1 meter from each other.

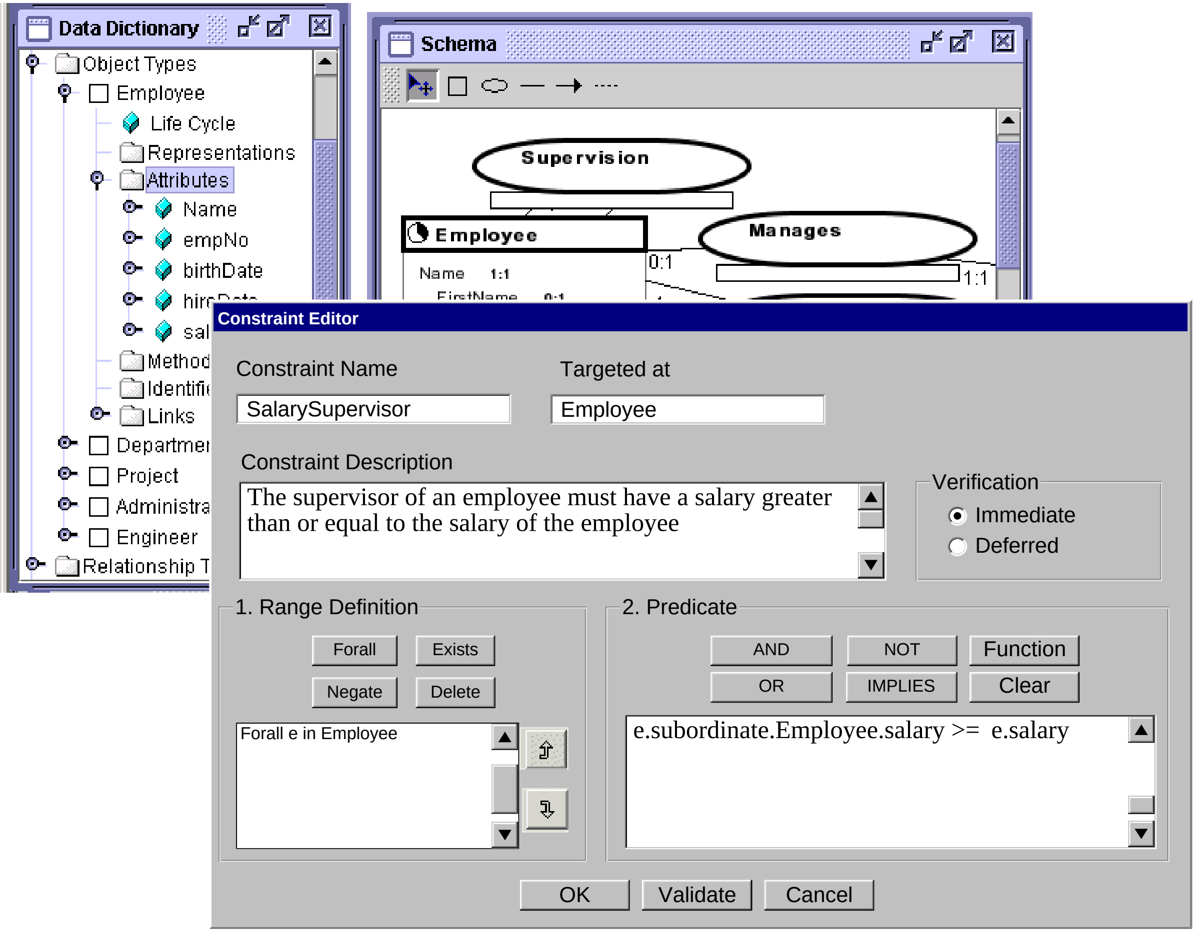
```
create or replace trigger LakeDepthDistance1m
before insert on Lake for each row
declare rowcnt number;
begin
  select count(*) into rowcnt
  from table(:new.depth) d1, table(:new.depth) d2
  where sdo_within_distance(d1.point,d2.point,'distance=1')='TRUE' )
  if rowcnt <> 0 then
    raise_application_error(-20403,
      'Points must be at least 1 m. from each other')
  end if;
end;
```

Conclusions

- Usual approach for information systems design induces **important semantic loss**
- We presented a methodology to ensure **semantic equivalence** between conceptual and physical schemas
- This requires **implicit** integrity constraints at the **logical and physical levels**
- We showed our methodology using the MADS conceptual model
- The methodology is **generic** and can be applied to **any conceptual model**
- Important issues to be addressed
  - **Scalability**: real-size application would generate 100s of constraints
    $\Rightarrow$ selection of which constraints will be implemented
  - **Optimization**: implication of constraints
    $\Rightarrow$ could lead to performance increase

Future Works: Explicit Integrity Constraints

- **Visual** specifications of the constraints at a **conceptual level**



- **Semi-automatic translation** of such constraints

Preserving Semantics when Transforming Conceptual Spatio-Temporal Schemas

Questions ?