Implementing Conceptual Spatio-Temporal Schemas in Object-Relational DBMSs

---

Esteban ZIMÁNYI, Mohammed MINOUT
Department of Computer & Decision Engineering (CoDE)
Université Libre de Bruxelles
{ezimanyi,mminout}@ulb.ac.be
http://code.ulb.ac.be

Semantic-based Geographical Information Systems (SeBGIS'06)
October 29, 2006

Contents

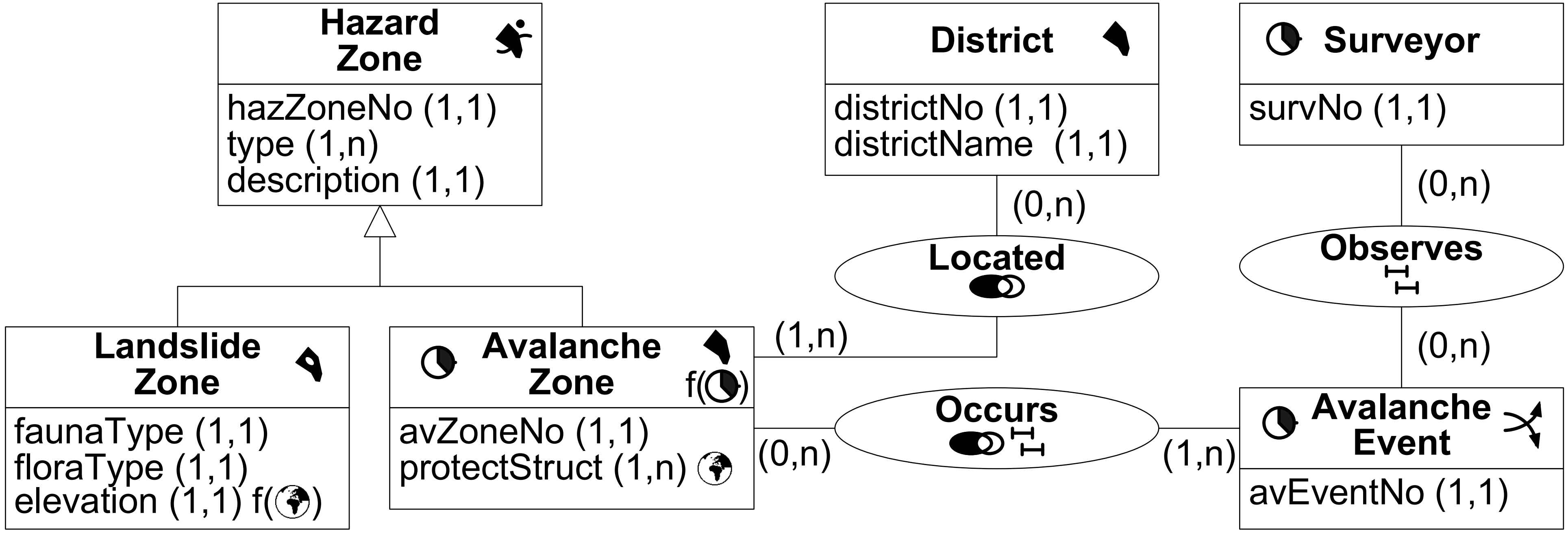
(Geographical) Information System Design

- 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, 2006, 476 pp.

The MADS Model: Example Conceptual Schema



Hazard Zone
hazZoneNo (1,1)
type (1,n)
description (1,1)

District
districtNo (1,1)
districtName (1,1)

Surveyor
survNo (1,1)

Landslide Zone
faunaType (1,1)
floraType (1,1)
elevation (1,1) f()

Avalanche Zone f()
avZoneNo (1,1)
protectStruct (1,n)

Avalanche Event
avEventNo (1,1)

Located
(0,n)
(1,n)

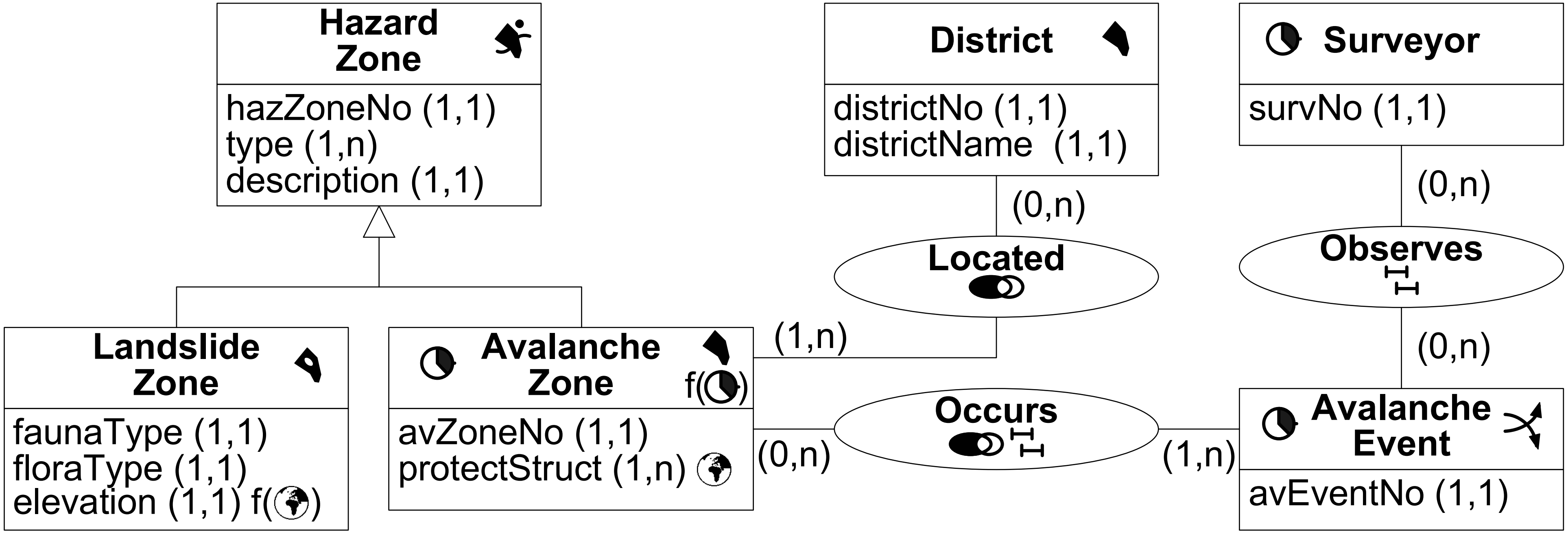
Observes
(0,n)
(0,n)

Occurs
(0,n)
(1,n)

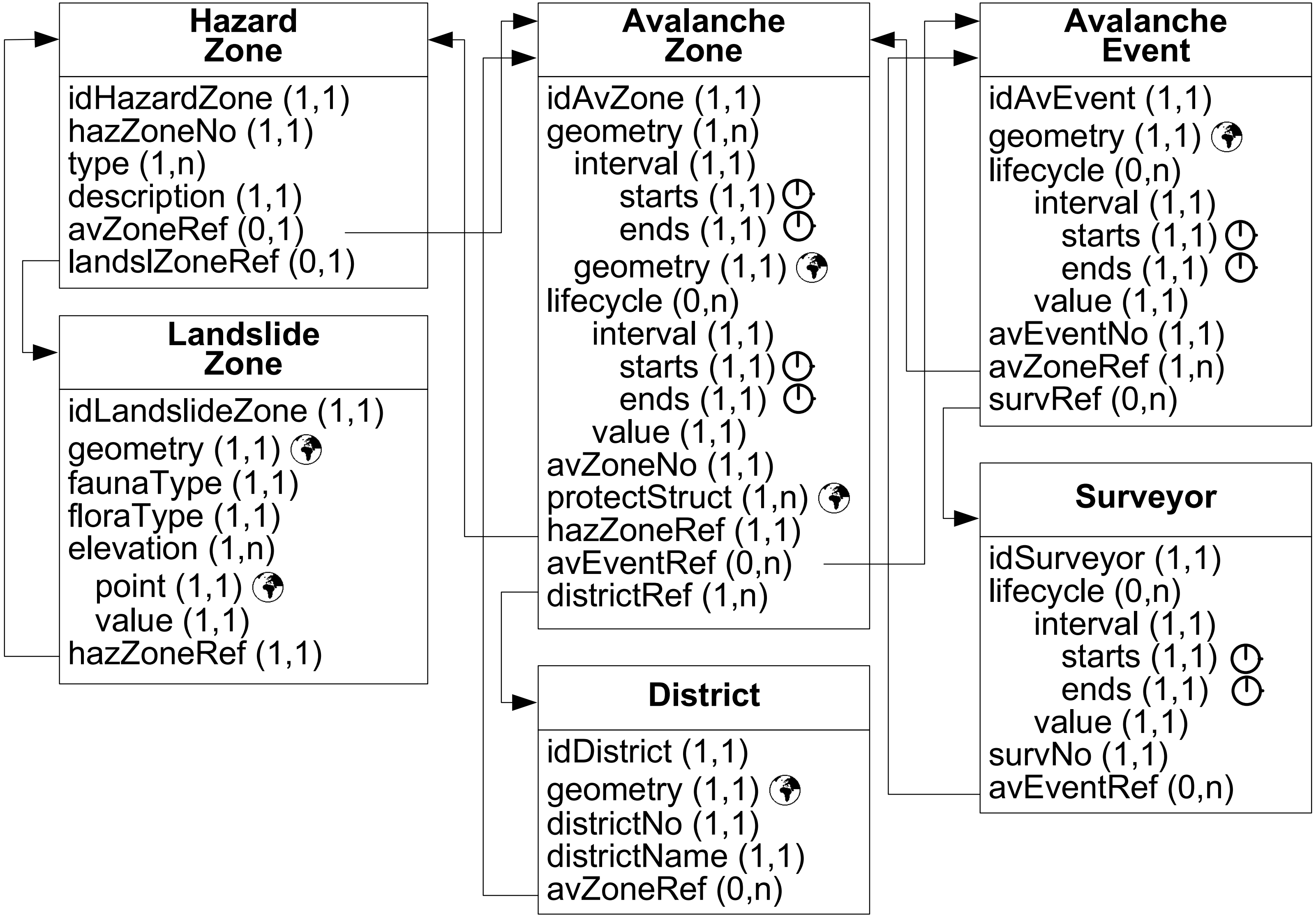
Translating MADS Schemas

- **Transformational approach**: replacing expressive MADS concept ⇒ set of constructs available in target implementation platform
  - Logical models: Relational, Object-Relational, ...
  - Physical models: Oracle, MapInfo, ArcInfo, ...
- **Spatial O/R types**:
  - materializing predefined `geometry` attribute
  - specialized spatial types ⇒ 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: OR Logical Schema



Hazard Zone

- idHazardZone (1,1)
- hazZoneNo (1,1)
- type (1,n)
- description (1,1)
- avZoneRef (0,1)
- landslZoneRef (0,1)

Landslide Zone

- idLandslideZone (1,1)
- geometry (1,1) 🌍
- faunaType (1,1)
- floraType (1,1)
- elevation (1,n)
  - point (1,1) 🌍
  - value (1,1)
- hazZoneRef (1,1)

Avalanche Zone

- idAvZone (1,1)
- geometry (1,n)
  - interval (1,1)
    - starts (1,1) ⏱
    - ends (1,1) ⏱
  - geometry (1,1) 🌍
- lifecycle (0,n)
  - interval (1,1)
    - starts (1,1) ⏱
    - ends (1,1) ⏱
  - value (1,1)
- avZoneNo (1,1)
- protectStruct (1,n) 🌍
- hazZoneRef (1,1)
- avEventRef (0,n)
- districtRef (1,n)

District

- idDistrict (1,1)
- geometry (1,1) 🌍
- districtNo (1,1)
- districtName (1,1)
- avZoneRef (0,n)

Avalanche Event

- idAvEvent (1,1)
- geometry (1,1) 🌍
- lifecycle (0,n)
  - interval (1,1)
    - starts (1,1) ⏱
    - ends (1,1) ⏱
  - value (1,1)
- avEventNo (1,1)
- avZoneRef (1,n)
- survRef (0,n)

Surveyor

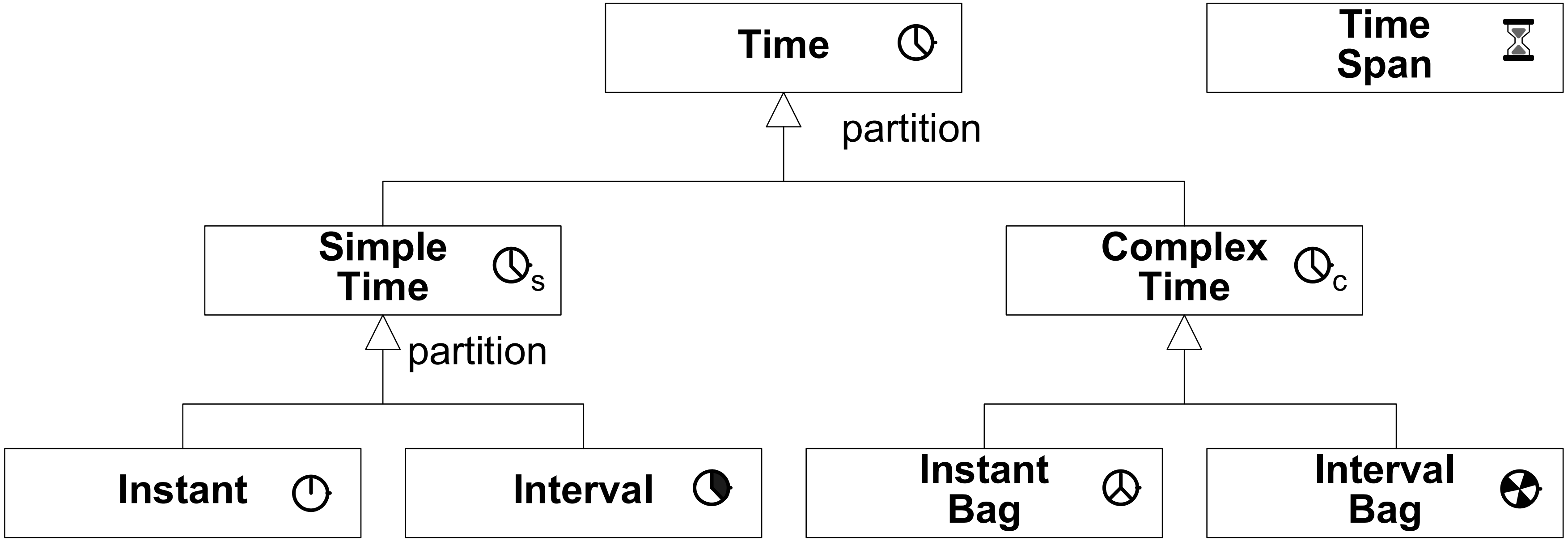
- idSurveyor (1,1)
- lifecycle (0,n)
  - interval (1,1)
    - starts (1,1) ⏱
    - ends (1,1) ⏱
  - value (1,1)
- survNo (1,1)
- avEventRef (0,n)

Translating MADS Schemas: Oracle Physical Schema

- Rewriting logical schema ⇒ schema expressed in language of target platform

```
create or replace type DId as object (idValue integer);
create or replace type DGeometrySet as table of mdsys.sdo_geometry;
create or replace type DAvEventSetRef as table of ref DAvalancheEvent;
create or replace type DAvZoneSetRef as table of ref DAvalancheZone;
create or replace type DSurvSetRef as table of ref DSurveyor;
create or replace type DAvalancheZone as object (idAvZone DId,
   geometry DTVGeometry, avZoneNo integer,
   protecStruct DGeometrySet, hazZoneRef ref DHazardZone,
   avEventRef DAvEventSetRef, districtRef DDistrictSetRef);
create or replace type DAvalancheEvent as object (idAvEvent DId,
   geometry mdsys.sdo_geometry, lifecycle DLifecycle,
   avEventNo integer, avZoneRef DAvZoneSetRef, survRef DSurvSetRef);
create table AvalancheZone of DAvalancheZone
    nested table geometry store as AvZoneGeometryNT
    nested table protectStruct store as AvZoneProtectStructNT;
    nested table avEventRef store as AvZoneAvEventRefNT
    nested table districtRef store as AvZoneDistrictRefNT;
...
```

Spatio-Temporal Data Types: MADS



Methods in Spatio-Temporal Data Types: MADS

`Time`

- `dimension()`: dimension (0 or 1) of temporal value
- `envelope()`: minimum bounding interval of temporal value
- `boundary()`: boundary of temporal value, an abstract method
- `connected()`: whether every couple of instants of temporal value are connected
- `duration()`: total time span of temporal value
- `meets(Time t)`, `intersects(Time t)`, `overlaps(Time t)`, . . .
- `intersection(Time t)`, `union(Time t)`, `difference(Time t)`, . . .

`Interval`

- **Additional** methods: `starts()` and `ends()`
- **Redefines** methods inherited from `Time`: `dimension`, `boundary`

Varying Data Types: MADS

- Allow to represent **data that vary on space and/or on time**
- Defined by a function
  - **domain**: any spatial and/or temporal extent
  - **range**: any set of values
- Some methods of varying data types
  - `defSpace()`, `defTime()`: spatial and temporal extent on which function is defined
  - `rangeValues()`: set of values taken by function (its range)
  - `atLocation(Geo g)`, `atTime(Time t)`: portion of function defined for the spatial/temporal extent given as parameter

Lifecyle Data Type: MADS

- Used for capturing **lifecycle of object and relationship types**
- Composed of
  - time-varying attribute `status`: status of instances
  - temporal attributes `dob` and `dod`: date of first activation (“date of birth”) and date of deactivation (“date of death”) of instances
- Some methods of the lifecycle data type
  - `lifeSpan()`: temporal extent during which instance existed
  - `activeSpan()`: temporal extent during which lifecyle is active
  - `status(Intant t)`: status value at instant given as parameter

Implementing Temporal Types in OR DBMSs: Oracle

- Implemented in a package that will be used by any database ⇒ independent of the schema
- Inheritance model in Oracle **does not allow to implement the hierarchy of types**
  - E.g., not allowed to define in `DTime` a method `envelope` returning a value of type `DInterval`, a subtype of `DTime`
- **Only the lower-level types** of the hierarchy are defined ⇒ all methods **must be repeated**

```
create or replace type DInstant as object ( instant Date,
  member function Tdimension() return number,
  member function envelope() return DInterval,
  member function intersection(t DTime) return DInterval,
  member function Tintersects(t DTime) return integer, ...
) instantiable final;
create or replace type DInterval as object (
  starts Date, ends Date,
  /* ... the same set of member functions as above ... */
) instantiable final;
...
```

Implementing Temporal Types in OR DBMSs: Oracle, cont.

```
create or replace type body DInterval as
  member function envelope return DInterval as
  begin
    return DInterval(self.starts,self.ends);
  end;
  member function Tintersects(t in DInterval) return integer as
  begin
    if (self.starts<t.ends and t.starts<self.ends) then returns 1;
    else returns 0;
    end if;
  end;
  member function intersection(t DInterval) return DInterval as
  begin
    if (self.Tintersects(t))
    then return
       DInterval(maxDate(self.starts,t.starts),minDate(self.ends,t.ends));
    else return null;
    end if;
  end;
  ...
end;
```

Implementing Time-Varying Types in OR DBMSs: Oracle

- Oracle **does not allow parameterized types**

  ⇒ a definition needed for each time-varying data type

- Example for time-varying geometries

```
create or replace type GeometrySet as table of mdsys.sdo_geometry;
create or replace type TVGeometryValue as object (
  interval DInterval, geometry mdsys.sdo_geometry );
create or replace type TVGeometryValues as table of TVGeometryValue;
create or replace type TVGeometry as object (
  tv_geometry TVGeometryValues,
  member function defTime return DInstant,
  member function defTime return DInterval,
  member function rangeValues return GeometrySet,
  member function atTime(t DInstant) return TVGeometry,
  member function atTime(t DInterval) return TVGeometry,
  ...
)
```

Implementing Time-Varying Types in OR DBMSs: Oracle

- Example of function definition: projecting the time-varying geometry on an interval

```
member function atTime(t DInterval) return TVGeometry is
  v1 TVGeometryValues := TVGeometryValues();
  cursor c_TVGeometryValues is
    select v.interval,v.geometry from table(self.tv_geometry) v;
begin
  for v2 in c_TVGeometryValues loop
    if (v2.interval.Tintersects(t)=1) then
      v1.extend;
      v1(v1.last):=
        TVGeometryValue(v2.interval.intersection(t),v2.geometry);
    end if;
  end loop;
  return TVGeometry(v1);
end;
```

Implementing the Lifecycle Type in OR DBMSs: Oracle

- Type definitions for lifecycle type

```
create or replace type DIntervalBag as table of DInterval;
create or replace type TVStatusValue as object (
  interval DInterval, status varchar2(9));
create or replace type TVStatusValues as table of TVStatusValue;
create or replace type DLifecycle as object (
  tv_status TVStatusValues, dob Date, dod Date,
  member function lifespan return DInterval,
  member function activespan return DInterval,
  member function status(d in Date) return varchar2(9),
  ...
)
```

Implementing the Lifecycle Type in OR DBMSs: Oracle

- Exemple member functions

```
member function activeSpan return DIntervalBag is
  v1 DIntervalBag := DIntervalBag();
  cursor c_TVStatusValues is
    select s.interval,s.status from table(self.tv_status) s;
begin
  for v2 in c_TVStatusValues loop
    if (v2.status=’active’) then
      v1.extend;
      v1(v1.last)= DInterval(v2.interval);
    end if;
  end loop;
  return v1;
end;
member function status(t DInterval) return varchar2(9) is
  v varchar2(9);
begin
  select s.status into v from table(self.tv_status) s
  where s.interval.contains(t)
  return v;
end;
```

Temporal Queries

- MADS algebra allows to use all methods and operators of spatial and temporal data types when formulating queries
- Find the geometry of avalanche zones between 01/05/2004 and 08/12/2005

```
select [ geometry.atTime([01/05/2004, 08/12/2005]) ] AvalancheZone
```

- Translation of queries must take into account
  - transformation of **conceptual schema** into **logical schema**
  - transformation of spatial/temporal **methods and operators**
- Example query

```
select a.geometry.atTime(DInterval(01/05/2004,08/12/2005))
from AvalancheZone a
```

Temporal Queries, cont.

- Find the identifiers of districts in which are located active avalanche zones whose geometry did not change between 22/05/2002 and 01/06/2006

```
select d.idDistrict
from district d, table(AvalancheZoneRef) refAv,
     table(refAv.column_value.lifecycle) l
where Tcontains(l.activespan(),sysdate) and
  ( select count(a.geometry.atTime(DInterval(22/05/2002,01/06/2006))
    from AvalancheZone a
    where refAv.column_value.idAvZone=a.idAvZone ) =< 1
```

Temporal Integrity Constraints

- Implementation of methods can be used for expressing integrity constraints
- Trigger raising an error upon insertion of avalanche zone instances if two intervals of the lifecycle overlap

```
create or replace trigger AvalancheZoneLCOverlappingIntervals
before insert on AvalancheZone for each row
begin
  if exists (
    select * from table(:new.lifecycle) l1, table(:new.lifecycle) l2
    where l2.Toverlaps(DInterval(l1.starts,l1.ends))
    then raise_application_error(-20300,
      'Overlapping intervals in lifecycle')
  end if;
end
```

Conclusions and Future Work

- **Conceptual** specifications must be **translated** into the **operational** model provided by GISs and DBMSs
- Traditionally, **transformation** realized by CASE tools **only copes with attributes**
- Translation must also include generation of **functions and procedures** that allow the **manipulation** of spatio-temporal information
- This **generation** can be **realized automatically** and showed an example using an object-relational DBMS
- Future work
  - Translation of spatio-temporal **queries and updates**
  - Translation of spatio-temporal **integrity constraints**: explicit and implicit
  - Support of **other implementation platforms**

Implementing Conceptual Spatio-Temporal Schemas in Object-Relational DBMSs

---

Questions ?