A Data Model and Data Structures for Moving Objects Databases

Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider
SIGMOD 2000, Dallas, TX

Presented by Martin Lund Kristiansen
October 22, 2007
Outline

- Motivation
- Abstract model
- Discrete model
- Conclusion
- Evaluation
Motivation

- Many applications require DBMSs to manage spatial objects
  - Countries, roads, power lines, etc.
- But also **moving** (temporal) objects
  - Airplanes, hurricanes, precipitation (nedbør)
- Dubbed “moving objects databases”
Motivation

- A previous article presented an **abstract model**
  - Focuses on essential concepts
  - But no representation details
- Now a **discrete model** is presented
  - Contains representation details
  - Implementable
Abstract model
Query example

- We can query on the relation:
  planes (airline: string, id: string, flight: moving(point))

- All flights of Lufthansa > 5000 km:
  SELECT airline, id
  FROM planes
  WHERE airline = “Lufthansa”
    AND length(trajectory(flight)) > 5000
Query example

- Relation is still:
  planes (airline: \textit{string}, id: \textit{string}, flight: \textit{moving(point)})

- All pairs of planes that came closer to each other than 500m:
  \begin{verbatim}
  SELECT p.airline, p.id, q.airline, q.id
  FROM planes p, planes q
  WHERE p.id != q.id
  AND val(initial(atmin(distance(p.flight, q.flight))))) < 0.5
  \end{verbatim}
Type Constructors

- Model specifies type system
- Data type examples:
  - `int`, `moving(point)`
  - `moving(point)` value is a function from time into `point` values

<table>
<thead>
<tr>
<th>Argument kind</th>
<th>Result kind</th>
<th>Type constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE</td>
<td>BASE</td>
<td><code>int, real, string, bool</code></td>
</tr>
<tr>
<td>SPATIAL</td>
<td>SPATIAL</td>
<td><code>point, points, line, region</code></td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
<td><code>instant</code></td>
</tr>
<tr>
<td>BASE u TIME</td>
<td>RANGE</td>
<td><code>range</code></td>
</tr>
<tr>
<td>BASE u SPATIAL</td>
<td>TEMPORAL</td>
<td><code>intime, moving</code></td>
</tr>
</tbody>
</table>
Operations

- Model also specifies operators
- Not in table:
  - atinstant, derivative, speed, and others...

<table>
<thead>
<tr>
<th>Operation</th>
<th>Argument kind</th>
<th>Result kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>trajectory</td>
<td>moving(point)</td>
<td>line</td>
</tr>
<tr>
<td>length</td>
<td>line</td>
<td>real</td>
</tr>
<tr>
<td>distance</td>
<td>moving(point) × moving(point)</td>
<td>moving(real)</td>
</tr>
<tr>
<td>atmin</td>
<td>moving(real)</td>
<td>moving(real)</td>
</tr>
<tr>
<td>initial</td>
<td>moving(real)</td>
<td>intime(real)</td>
</tr>
<tr>
<td>val</td>
<td>intime(real)</td>
<td>real</td>
</tr>
</tbody>
</table>
Discrete model
Discrete model

- Defines domains for the data types in the abstract model
- Represents only a subset of the values of the corresponding abstract model
- All abstract type constructors have discrete counterparts, except for the `moving` constructor
**Type Constructors**

- UNIT type is introduced to explicitly support temporal types.
- We therefore distinguish between e.g. a non-temporal *real* and its temporal counterpart *ureal*.

<table>
<thead>
<tr>
<th>Argument kind</th>
<th>Result kind</th>
<th>Type constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>→ BASE</td>
<td>BASE</td>
<td>int, real, string, bool</td>
</tr>
<tr>
<td>→ SPATIAL</td>
<td>SPATIAL</td>
<td>point, points, line, region</td>
</tr>
<tr>
<td>→ TIME</td>
<td>TIME</td>
<td>instant</td>
</tr>
<tr>
<td>BASE u TIME</td>
<td>RANGE</td>
<td>range</td>
</tr>
<tr>
<td>BASE u SPATIAL</td>
<td>UNIT</td>
<td>const</td>
</tr>
<tr>
<td></td>
<td>UNIT</td>
<td>ureal, upoint, upoints, uline, uregion</td>
</tr>
<tr>
<td>UNIT</td>
<td>MAPPING</td>
<td>mapping</td>
</tr>
</tbody>
</table>
Sliced Representation

- Built by `mapping` constructor
  - e.g. `mapping(point)`
- Each slice consists of a UNIT type `(i, v)`
  - `i` is a time interval
  - `v` is a simple function
- For discrete-only values `const` constructor is used (e.g. “moving” `int`, `bool`)

![Diagram showing sliced representation](image)
Abstract & Discrete Temporal Types

- Discretely changing values use the `const` constructor
- Continuously changing values use special UNIT types

<table>
<thead>
<tr>
<th>Abstract Type</th>
<th>Discrete Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>moving(int)</code></td>
<td><code>mapping(const(int))</code></td>
</tr>
<tr>
<td><code>moving(bool)</code></td>
<td><code>mapping(const(bool))</code></td>
</tr>
<tr>
<td><code>moving(real)</code></td>
<td><code>mapping(ureal)</code></td>
</tr>
<tr>
<td><code>moving(point)</code></td>
<td><code>mapping(upoint)</code></td>
</tr>
<tr>
<td><code>moving(points)</code></td>
<td><code>mapping(upoints)</code></td>
</tr>
</tbody>
</table>
Basic Data Type Domains

- Base types and time type:

\[ D\text{int} = \text{int} \cup \{\bot\} \]
\[ D\text{string} = \text{string} \cup \{\bot\} \]
\[ D\text{instant} = \text{Instant} \cup \{\bot\} \]
  \( (\text{Instant} = \text{real}) \)
\[ D\text{real} = \text{real} \cup \{\bot\} \]
\[ D\text{bool} = \text{bool} \cup \{\bot\} \]

- Spatial data types:

\[ D\text{point} = \text{Point} \cup \{\bot\} \]
\( (\text{Point} = \text{real} \times \text{real}) \)
\[ D\text{points} = \mathcal{P}(\text{Point}) \]

- .
- .
- .
Line Data Type Domain

\[ \text{Seg} = \{(u, v) | u, v \in \text{Point}, u < v \} \]

\[ D_{\text{line}} = \{ S \subseteq \text{Seg} | \forall s, t \in \text{Seg} : s \neq t \land \text{collinear}(s, t) \Rightarrow \text{disjoint}(s, t) \} \]
Region Data Type Domain

- Segments used to form polygons
- Again, approximation is used
- May result in false positives on e.g. joins (rain example)
- Is formally defined in the paper
Units

- A unit/slice is defined by:
  \[ \text{Unit}(S) = \text{Interval}(\text{Instant}) \times S \]
  - First component is the *unit interval*, second component the *unit function*
  - The function component maps a unit function for a given instant of time into a value
  \[ \iota_\alpha : S_\alpha \times \text{Instant} \rightarrow D_\alpha \]
Moving Point Data Type Domain

- Moving point is type *upoint* (UNIT type)
- A linearly moving point is described by:

\[
\nu((x_0, x_1, y_0, y_1), t) = (x_0 + x_1 \cdot t, y_0 + y_1 \cdot t) \quad \forall t \in \text{Instant}
\]

\[
D_{\text{upoint}} = \text{Interval}(\text{Instant}) \times \text{MPoint}
\]

\[
\text{MPoint} = \{(x_0, x_1, y_0, y_1) | x_0, x_1, y_0, y_1 \in \text{real}\}
\]
Moving Lines & Regions

- Linear approximation is used again
- Definitions for domains of moving lines and regions can be found in the article
Conclusion

- Discrete model implements all data types of the abstract model
- Data structures explained for an example implementation
- Devised two algorithms for operations on discrete data structures
  - at instant and inside
Evaluation

- Positive
  - Well-written
  - Very concise and comprehensive

- Negative
  - Very complex—tries to do A LOT!
  - Builds on previous work
  - Error in SQL statement
  - As far as I know, there is no working data blade implementation
That’s it!