Formal Information Modeling for Standardization in the Spatial Domain

adapted by Damir Medak
credits to Andrew U. Frank

Overview

- Standardization for GIS
- Open GIS standards
- Problems with current standardization methods
- Try to use Functional Programming
- Conclusions

Background: Standards for GIS

- GIS is promising reuse of data collected by others – "sharing"
- Data must be transferred.
- Standardization of transfer to avoid a multiplicity of data transfer programs.

Standards

- Data Transfer Standards:
- USA and Germany: starting in 1970s
- SDTS (Spatial Data Transfer Standard USA)
  - Elaboration from 1980 to 1992
- CEN (Centre Européen de Normalisation)
- A similar European standard – available as draft

Industry standards

- The major vendors have defined formats which work and are widely used.
- AutoCAD, Intergraph, ESRI
- The elaboration of national standards, especially for small countries is a questionable effort.

Open GIS Consortium

- Geographic Data on the web does not require transfer of datasets, but access to the 'live' data.
- Open GIS : open with respect to vendors, proprietary formats
  - avoiding of 'locking in' of organizations
- Elaboration of industry standards: the topics of importance to industry at the time they are necessary.

Standardization of Interoperability

- Not standardization of data transfer.
- Description of operations which can be executed on another machine, including the interpretation of the results.
- Specification of operations – not description of data!

Open GIS Consortium Standards

- Subdivision of the field of GIS in several 'volumes' which are worked on.
- Completed first: feature geometry
- Currently 16 volumes
- A very substantial body of material!

Structure of a volume:

- Essential model – English description of what operations are considered, in what context and what they should produce:
- Ca. 1 page per volume

Abstract Specification

- An abstract description of the operations with their data types and what they do.
- Uses very often mathematical definitions, but is mostly (restricted) English.
- Typically 30 pages per volume

Implementation Specifications

- Detailed description of the implementation requirements for certain platforms of interoperability.
- Very detailed –up to 300 pages per volume

Problems with this approach

- This approach is 'state of the art' and similar to other standardization bodies.

Issues:

- Inconsistencies – definitions are not the same everywhere
- Contradictions –
- Holes: some important points are not described.
- Abstract and implementation specifications do not agree

Human check of the specifications

- Only humans check the specifications.
- Tools used are mostly for preparing the graphics.

Errors are costly:

- Inconsistencies in the standard require ‘thinking’ during the implementation.
- Resulting implementation may not be interoperable and long discussions on who is at fault and has to change result.

An Example:

- Coverage
- Datasets which define values for any points (within a limited region)

Examples:

- Digital elevation model,
- Cadastral partition of space
- Remote sensing image

Inconsistencies

- It is easy to find inconsistencies:
  - Between abstract model and implementation spec.
  - (Geometry is a supertype of Point…)
  - The implementation gives more operations than the abstract spec – does lead to inconsistencies across platforms.
- Need: derived implementation specifications from abstract specifications.

Algebraic Specifications

- Algebras allow specification without relying on previous understanding.
- (see the paper Frank & Kuhn 1995)
- Using a functional programming language

Translation of abstract specifications to Haskell

- Class in specifications becomes algebras (class in Haskell).
- Operations become functions (with all parameters)
- A very simple implementation is described as a model. It uses only lists of data elements.

More Abstraction

- The algebra can be described simply in terms of operations and thus become independent of the model.
- The model is important:
  - to create the specification and understand it.
  - to be able to execute the specifications.
- The abstraction is important:
  - to demonstrate that the specification is independent of the model.

Correctness

- Inconsistencies are discovered, can be discussed.
- Is this specifying what we wanted?
- Have domain experts read the (short) code – difficult.
- Have domain experts produce test cases and check the results – specifications in functional language are executable!

Outlook

- More tools necessary:
  - For example a tool to translate Haskell into UML and diagrams (software engineers love diagrams!)
  - Translate Haskell to readable C++ (probably possible, but not likely an attractive target)
  - Tools to combine text and Haskell code (using Word, not LaTEX)

Impediments

- Software engineers are
  - educated in imperative programming language
    - Unlearning is very difficult.
  - Concerned with execution speed (irrelevant for standardization!)
    - Writing non-optimized code is very difficult (emotionally)

However, changes occur:

  - Java takes over from C++ as the industry language
  - Object-orientation has superseded procedural abstraction.
- Changes are slow!

Object oriented modeling for GIS

- Lacks of relational database management systems
- What is object orientation?
- Objects as algebras
- Abstraction mechanisms
- Modeling behavior

What RDBMS do not provide?

- sophisticated treatment of real-world geometry
- representation of the same data at different conceptual levels of resolution and detail
- management of history and versions of objects
- combinations of measurements of different resolution and accuracy

Object-orientation

- a design method that focuses on modeling objects as humans perceive them in reality
- it combines modeling of the structure and the behavior of the objects
- corresponds with multi-sorted algebras

Description of an object consist of:

- a name for its type
- a set of operations which are applicable to objects of this type
- a set of axioms in terms of the operations

Object-oriented abstraction mechanisms

- classification: mapping of several objects (instances) onto a common class
- generalization: groups several classes of objects with common operations into a more general superclass
- association: relates two or more indpendent objects (member-of, neighborhood)
- aggregation: models objects which consist of other objects

Modeling behavior

- Inheritance is a method to define a class in terms of one or more other, more general classes. Properties which are common to a class and its subclasses are defined only once and inherited to all objects of the subclass
- Propagation describes how a *value* of a property of one class is derived from values of properties of another class.

Seven-steps method

1. Candidate classes
2. Define classes
3. Establish associations
4. Expand many-to-many associations
5. Attributes
6. Normalization
7. Operations (behavior)

1. Candidate classes

- a *thing* (entity) i represented in English as a *noun*
- a list of *nouns* that *might* name classes we are interested in
- four ways to generate a list:
  - client interviews
  - requirements model and other documentation
  - brainstorming (generating ideas)
  - the Delphi method (brainstorming by mail)

2. Define classes

- checking the importance of candidate classes
- three checks:
  - real-world identifier
  - definition
  - sample attributes and behaviors
- objects have identity
- if a class matters, its instances must have identity

3. Establish associations

- association is a relationship expressing the interactions between instances of two classes, represented by the verb that describes what they do to each other
- discovering the verb
- capturing all possible associations (matrix method)
- establishing the multiplicity (the number of instances of each class that can participate in an occurence of an association)
- Sentence = Subject + Verb + Number + Object

  An owner owns many Parcels
- difference between identifiable and quantifiable objects

4. Expand many-to many associations

- 1:M associations are simple to implement (foreign keys in RDBMS or pointers in OODBMS)
- M:M associations (a parcel owned by several owners, an owner having several parcels)
- placing event data attributes
- solution: intersection (associative) class
- every M:M association breaks out into a pair of 1:M associations with the M-ends pointing at this new intersection class

5. Attributes

- a lot of work to do, but simple concept
- for each class: list all the attributes the users can think of
- in larger models boxes will be cluttered – record just the key attributes
- use dictionaries to provide good definitions

6. Normalization

- Normalization is a proces of ensuring that every attribute is attached to the class of objects that it truly describes.
- primarily a database design technique, but valuable at all stages in systems development
- 1NF – no repeating fields
- 2NF – *all* the nonkey attributes are *fully* functionally dependent on the key
- 3NF – there are no transitive functional dependencies
- a list-valued attribute violates all normal forms
- denormalizing reduce complexity and improve response times, but makes it more difficult to extract the data later

7. Operations (Behavior)

- State of an object is defined by the set of values currently held by its attributes. A state has:
  - name
  - entry and exit actions
  - deferred events
  - internal transitions
- state transition: changing from one state to another

7. Operations (Behavior) cont.

- A transition is made up of:
  - source state (the state before the transition occured)
  - event trigger (event that causes the change)
  - guard condition (a Boolean expression that will prevent the transition if it evaluates to False)
  - action (an atomic operation that will be executed during the transition
  - target or destination state (the state after the transition is done)
- Internal transition is one that triggers actions or activities without causing the object to change state

7. Operations (Behavior) cont.

- statechart diagram: a map showing all permissible states and permissible transitions for a class of objects, along with events that cause these transitions and the actions and activities that result from them
- object life cycle consists of the various states an object may transition through, the set of permissible transitions and sequencing of those states and transitions, as it progresses from its initial creation to its eventual disposal

Geographic information – Methodology for feature cataloguing (ISO/DIS 19110)

- “functional language - programming language in which abstract data types are defined in terms of operations on the types, and in which algebraic axioms specify the results of each of the operations for each of the types.

  NOTE: In a functional language, feature types may be represented as abstract data types”
- “The use of functional language specifications to help define feature types is recommended.” (ISO/TC:211, 2001)