deegree 3

Designing a generic and efficient Feature and Object Model

lat/lon raumbezogene informationssysteme

Markus Schneider
schneider@lat-lon.de
http://www.lat-lon.de/

Clarification: *Feature and Object Model*

- Set of Java-classes and techniques that enable deegree to represent and persist geometric data objects
- Common types of geometric data objects
  - GML Features
    - Not all features adhere to an object/property structure
  - Catalogue datasets
    - Dublincore
    - ISO 19115/19119
    - EB-RIM
    - ...
  - Maybe others... suggestions?
- Usually defined by XML Schema documents

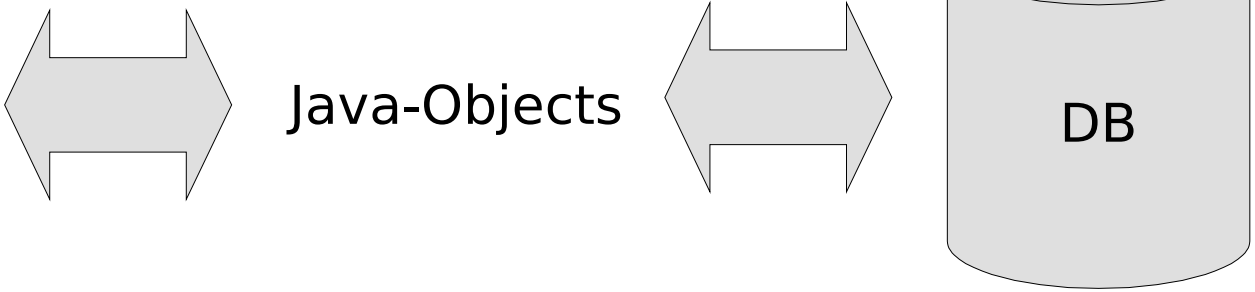
Example: GML feature object

```
<app:City gml:id="CITY_1">
  <app:name>Bonn</app:name>
  <app:center>
    <gml:Point srsName="EPSG:4326">...</gml:Point>>
  </app:center>
  <app:inCountry>
    <app:Country gml:id="COUNTRY_1">
      <app:name>Germany</app:name>
      <app:geom>
        <gml:MultiSurface srsName="EPSG:4326">...</gml:MultiSurface>
      </app:geom>
    </app:Country>
  </app:inCountry>
</app:City>
```

Primary functional requirements

1. XML-Marshalling/Unmarshalling
2. In memory-representation/processing
   - Query (in-memory) object collections using Filter expressions (e.g. visualization of features in WMS)
3. Persistence
   - Store objects in (relational) database
   - Query stored objects using Filter expressions

```
<?xml..
<FeatureCollection...
  <featureMember>
    <Country...
      <name...
    ...
  </featureMember>
</FeatureCollection/>
```



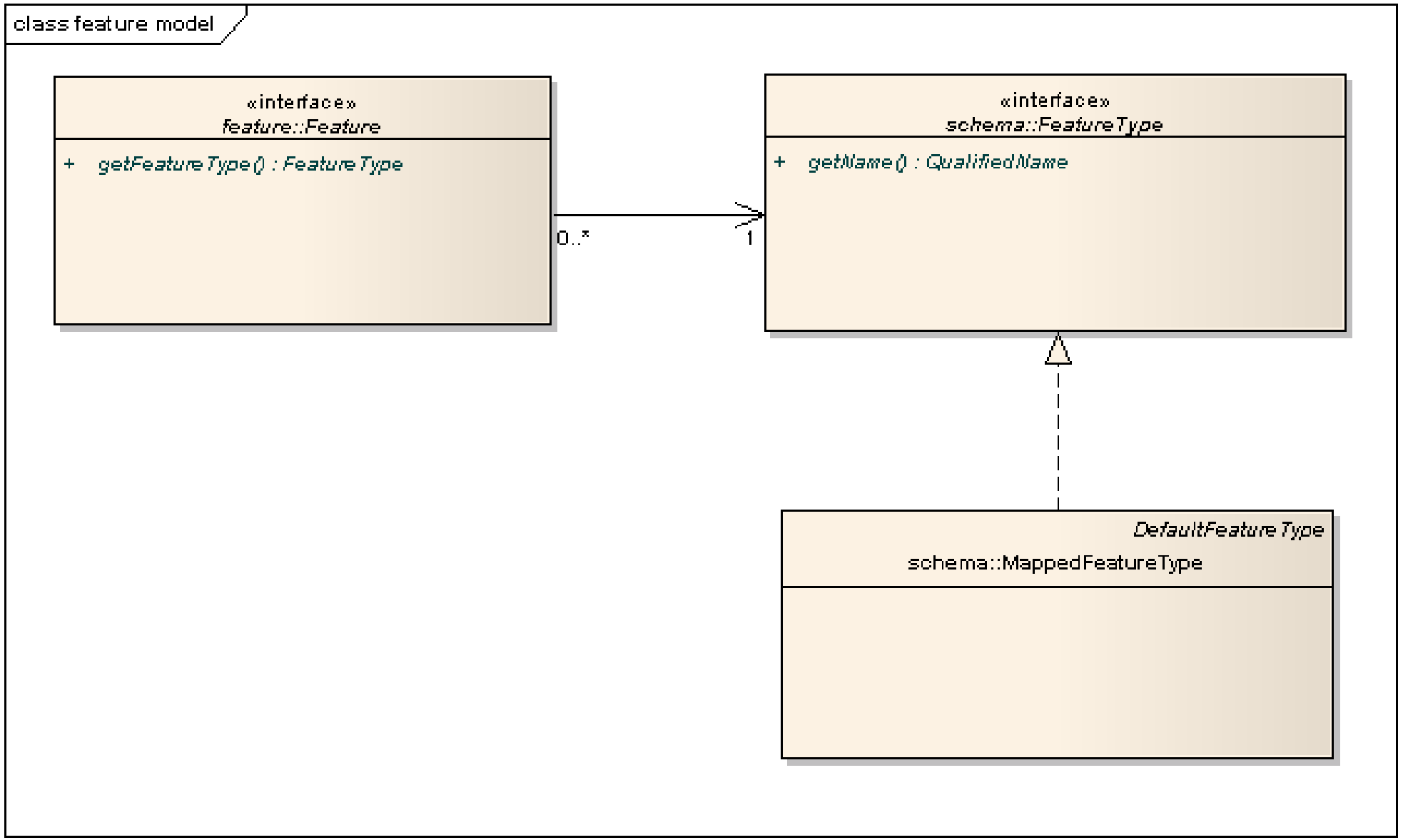
deegree 2 only supports GML features

- Feature type definition using GML application schemas
- Represent feature instances and feature types as Java objects (package `org.deegree.model.feature`)
- Convert between XML and Java Bean representation
- Traverse/change the feature/property structure
- Filter collections of feature objects using CQL (Filter expressions)

Persistence layer (datastore)

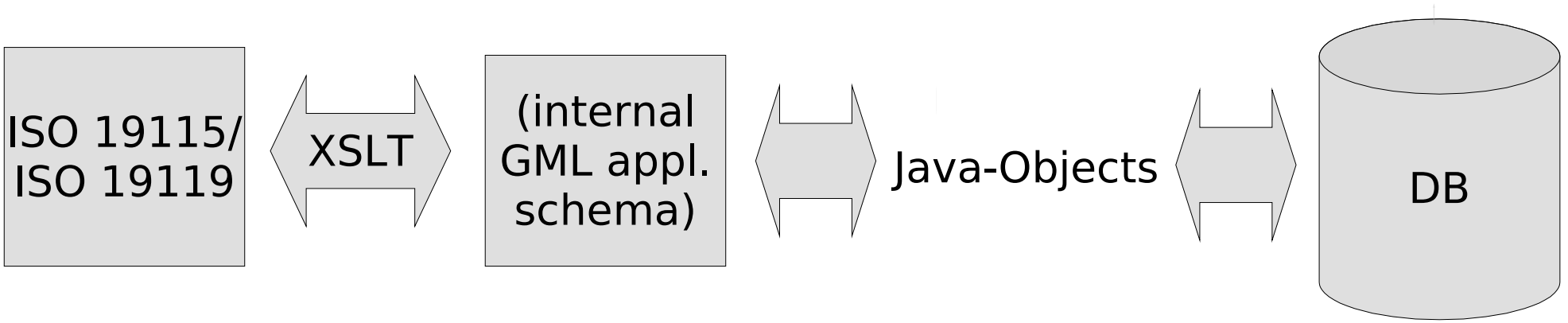
- Store, delete and update features in an SQL-database
- Query stored features using CQL-expressions
- Define XML-relational mapping using schema annotations

org.deegree.model.feature (deegree 2)



Handling non-feature objects in deegree 2

- Object schema is mapped to „internal“ GML schema
- Original XML structures fitted to features/properties using XSLT
  - Slow + resource-intensive
  - Error-prone
- Example: Implemenation of ISO Catalogue Service



Main shortcomings of current feature API

- Native support for GML features only, but we have similar processing needs for non-feature XML structures:
  - Catalogue datasets
  - GML application schemas that use complex properties (without adhering to the object/property – structure).
- Originally designed for simple features (w/o nested properties) and extended later
  - Does not support all GML core schema constructs (e.g. gml:AngleType with „uom“-attribute).
  - Inconsistent: feature collections are not features
  - Needs rewrite
- Problematic handling of type information (`Types.java`)
  - XML <-> Java <-> SQL

Goals for the deegree 3 object model

- Primary goal: Native support for arbitrary XML structures
  - Process and persist arbitrary XML objects without the need for XSLT-processing
- Secondary goal: customizable mapping of objects
  - Map simple features to an optimized Java representation
    - Improve memory requirements
  - Optionally support type-safe mapping: Use objects of class „Philosopher“ to represent „Philosopher“-feature instances
  - Generate custom classes from XML application schema automatically

Can we adopt existing technologies?

- Java-XML-Binding technologies have come a long way
  - JAXB 2.0 is part of JDK 1.6
  - Complex schemas compile (almost) out-of-the box
    - Philosopher.xsd
    - IMRO2008.xsd
- Does JAXB meet our requirements?
  - Problem: Schema information is converted to classes -> schema changes mean recompilation
  - JAXB addresses XML-Binding only, not persistence
    - Hibernate
    - HyperJAXB
    - What about support for spatial operators?

Design issues for deegree 3

- Base the generic object model upon existing feature model?
- Incorporate available XML-Binding frameworks?
  - JAXB, XMLBeans, Castor, ...
- Incorporate available persistence frameworks?
  - Hibernate, HyperJAXB, ..
- How do we represent type information?
  - XML type : Java class
  - XML type : Java object

Discussion summary 1/2

- It became clear that a strong connection between the deegree object model and XML schema exists
- Existing technologies like JAXB provide out-of-the box solutions for binding XML data to Java objects, but
  - Adding new functionality is difficult
  - Type information is converted to Java classes, which means that the use of new object types requires a compilation phase
- Developing a new XML schema framework provides a maximum of flexibility, but
  - XML schema is very complex
  - Will take a long time until a stable and feature-complete codebase exists

Discussion summary 2/2

- Similar aspects apply to object-relational mapping
- Existing technologies like Hibernate provide out-of-the box solutions, but
  - Not designed for spatial data persistence
  - Not designed for XML structures, but for Java classes
  - Flexibility is limited
- Maintaining and extending the deegree 2 datastore layer provides a maximum of flexibility, but
  - Needs rework to cope with arbitrary XML structures