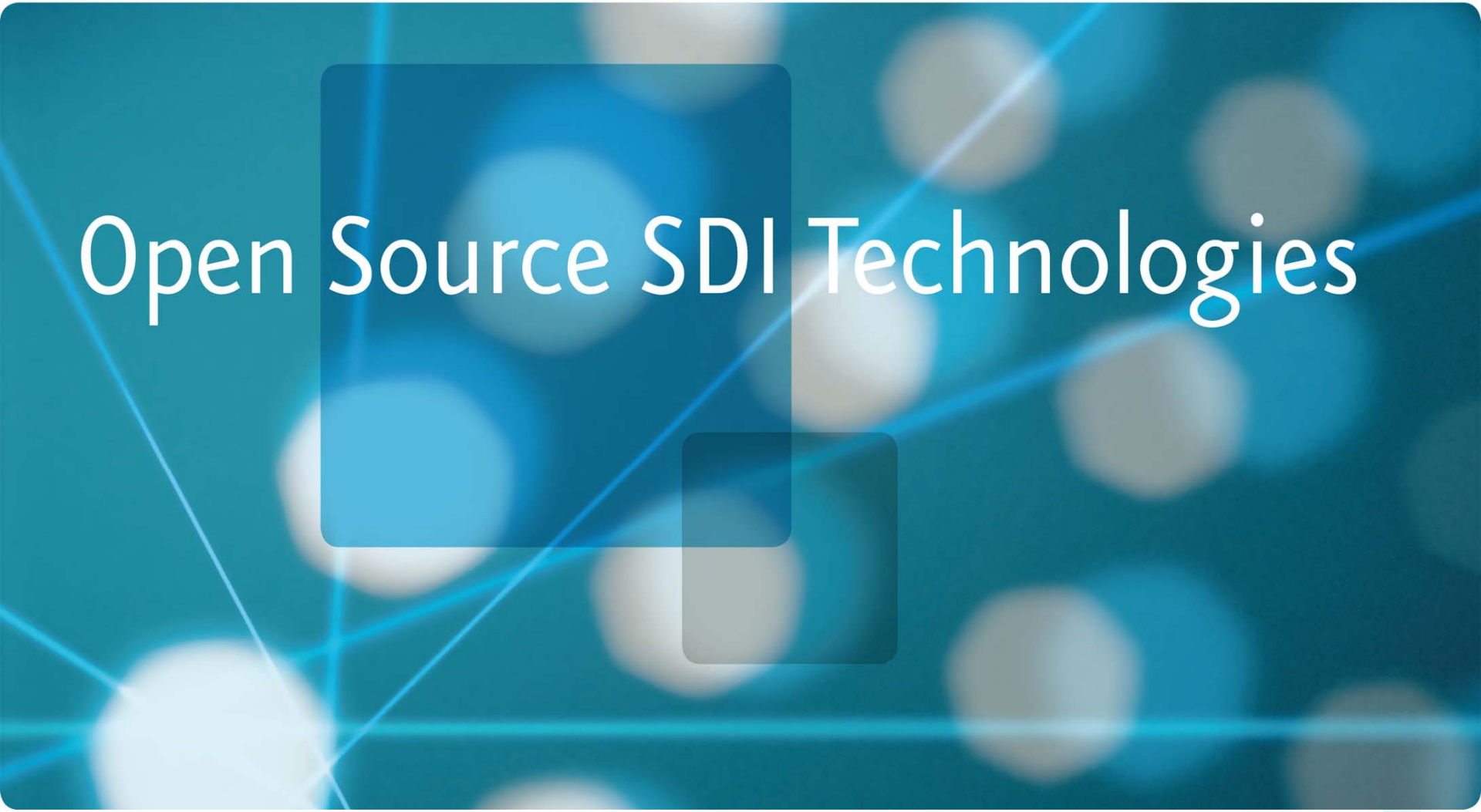
Sensor Observation Service

52north
exploring horizons



Johannes Echterhoff

Outline

- Overview
- Architecture
- Database
- Extending the service
- Installation

52north exploring horizons

Overview

General overview

- (soon) compliant to SOS 1.0 (OGC 06-009r6) – currently 06-009r5 is implemented
- Supported operations:
  - Core Profile: GetCapabilities, DescribeSensor, GetObservation
  - GetFeatureOfInterest
  - GetResult
  - GetObservationById
  - Soon to come: transactional profile (RegisterSensor, InsertObservation)
- Uses:
  - PostgreSQL+PostGIS
  - XmlBeans
  - JTS Topology Suite

Features

- extensible architecture → additional operations possible
- Data Access Objects → integration of other data sources
- filter: spatial, temporal and scalar
- different response types for different types of observations
- GZIP compression of responses
- 52n SOS Feeder Framework for inserting of data

Filter capabilities

- Spatial:
  - GeometryOperand: Envelope, Polygon, Point, LineString
  - SpatialOperator: BBOX, Contains, Intersects, Overlaps
- Temporal:
  - TemporalOperand: TimeInstant, TimePeriod
  - TemporalOperator: After, Before, During, Equals
- Scalar:
  - ComparisonOperator
    - =, != for all values
    - <,>,<=,>= for numeric values
  - no logical or arithmetic operators implemented right now

52north exploring horizons

Supported O&M types

- (generic) Observation (uses SweCommon)
- Measurement
  - For numeric values
- CategoryObservation
  - For categorical values

Capabilities cache

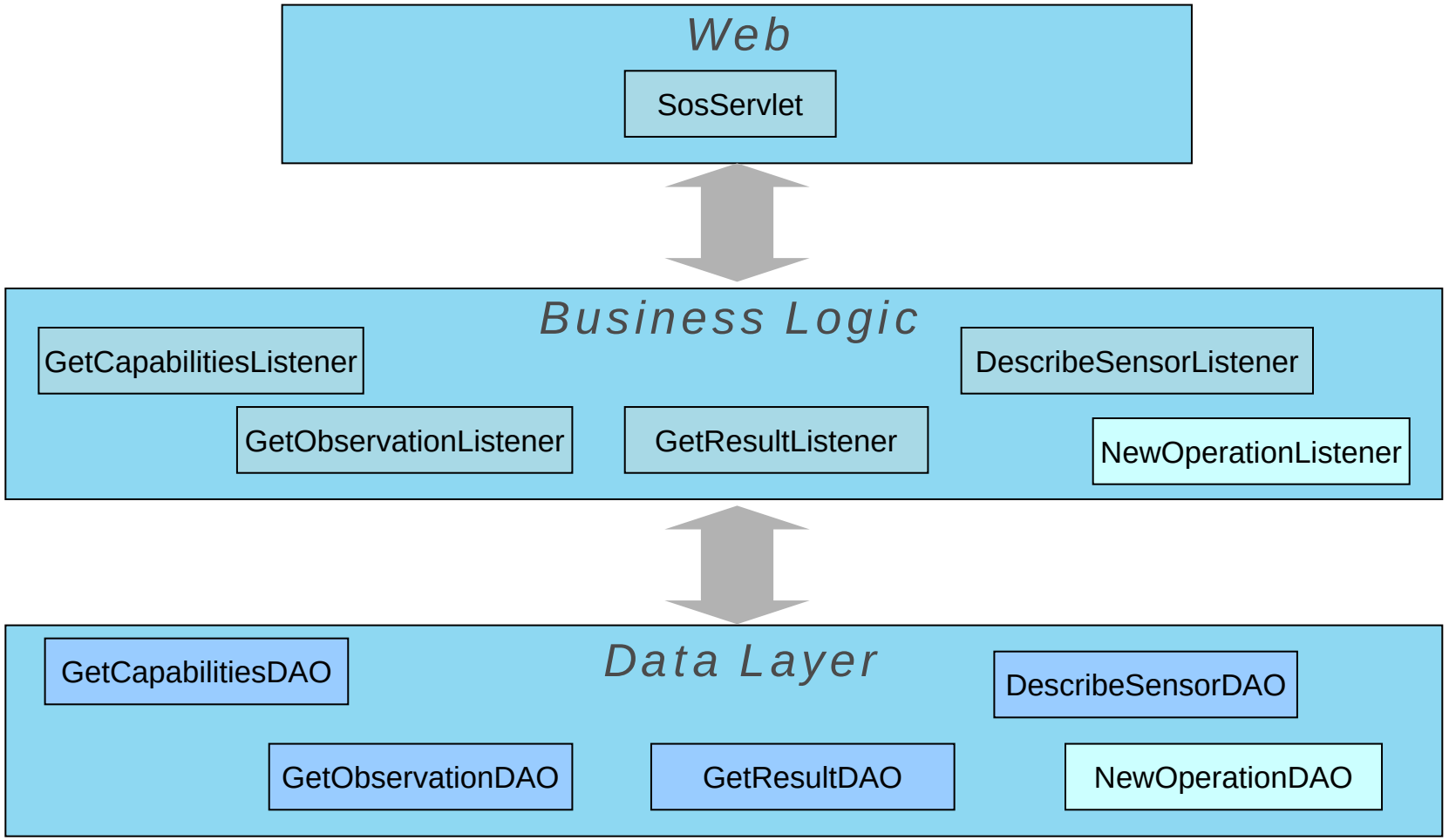
- New observations are inserted very often
- For stationary insitu sensors, features, procedures and phenomena do not change very often
- Metadata for observation offering is stored in `org.n52.sos.CapabilitiesCache` class to enhance performance for `GetCapabilities` operation
- If new features, procedures, phenomena or offerings are inserted or deleted, invoke `http://yourURLtoSOS/sos?Request=RefreshMetadata` to refresh metadata (HTTP GET request) → this is done by the 52north SosFeeder framework automatically

Composite phenomena

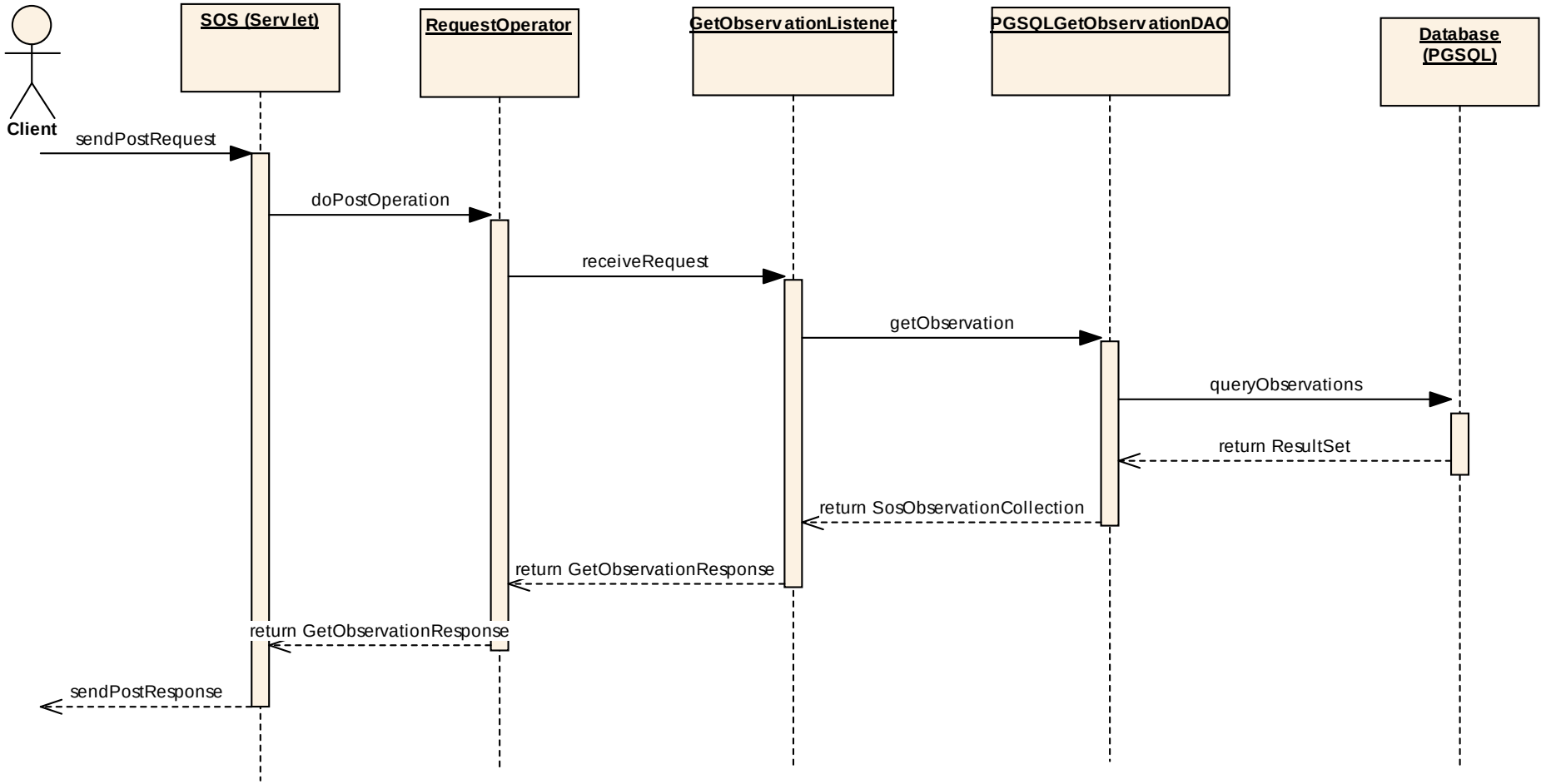
- several phenomena can be composed to one phenomenon: `swe:CompositePhenomenon`
- `CompositePhenomenon` consists of one or more single phenomena
- Example: Phenomenon `location` constists of values for `xcoord`, `ycoord` and `zcoord` – each being a single phenomenon
- Generic Observation type is used for composite phenomenon

Architecture

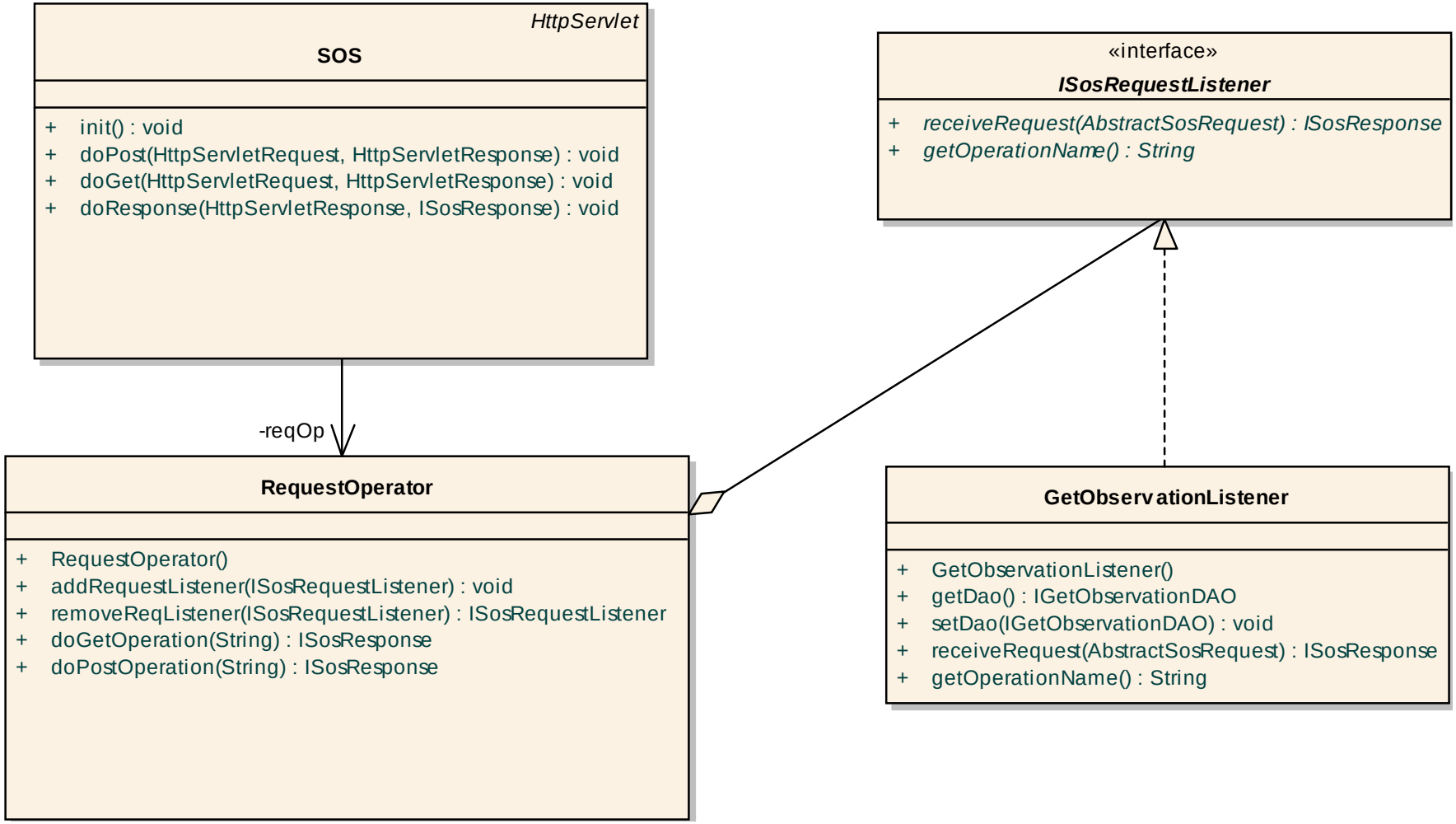
3-Tier Web Architecture



SOS – GetObservation sequence



SOS web/business tier



52north exploring horizons

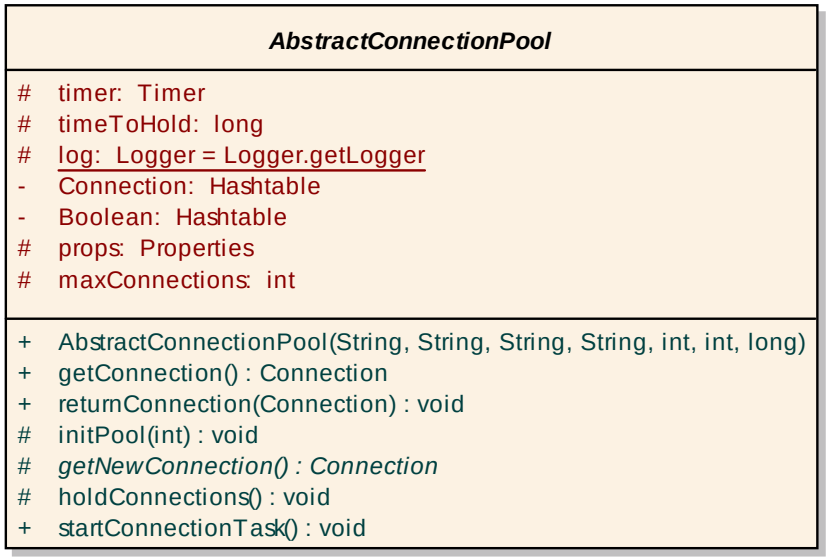
SOS web/business tier

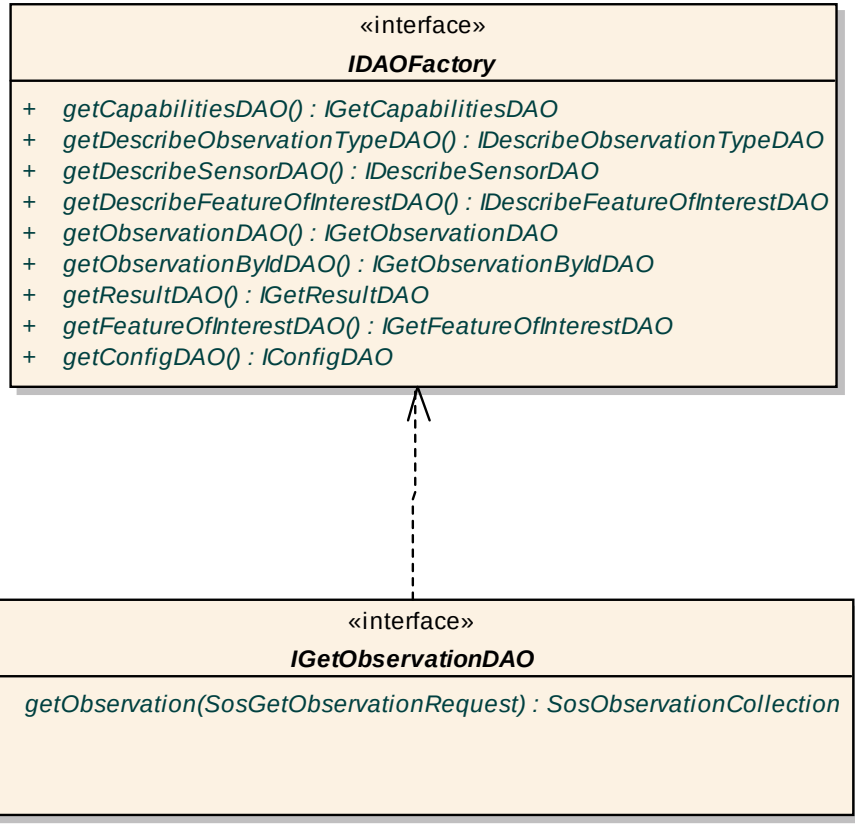
- Listener to use defined in `sos.config`
- Listeners loaded using reflection in `org.n52.sos.SosConfigurator`
- `org.n52.sos.SosConfigurator` reads in config files + initializes all dynamically loaded classes
- `org.n52.sos.RequestOperator` relates operation names to listeners, forwards requests

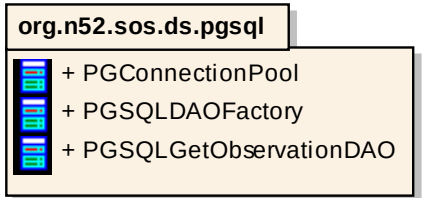
Request parsing and O&M encoding

- SOS uses XmlBeans for parsing/encoding
- Package `org.n52.sos.decode` contains decoder for parsing of requests
- Package `org.n52.sos.encode` contains encoder for O&M
- Internally SOS uses own O&M model (`org.n52.sos.ogc.om`) and own request objects (`org.n52.sos.request`)
  - Handling of substitution groups difficult with XmlBeans
  - Schema are very generic
- Package `org.n52.sos.response` contains response classes

Data Layer


AbstractConnectionPool
timer: Timer
timeToHold: long
log: Logger = Logger.getLogger
- Connection: Hashtable
- Boolean: Hashtable
props: Properties
maxConnections: int
+ AbstractConnectionPool(String, String, String, String, int, int, long)
+ getConnection() : Connection
+ returnConnection(Connection) : void
# initPool(int) : void
# getNewConnection() : Connection
# holdConnections() : void
+ startConnectionTask() : void

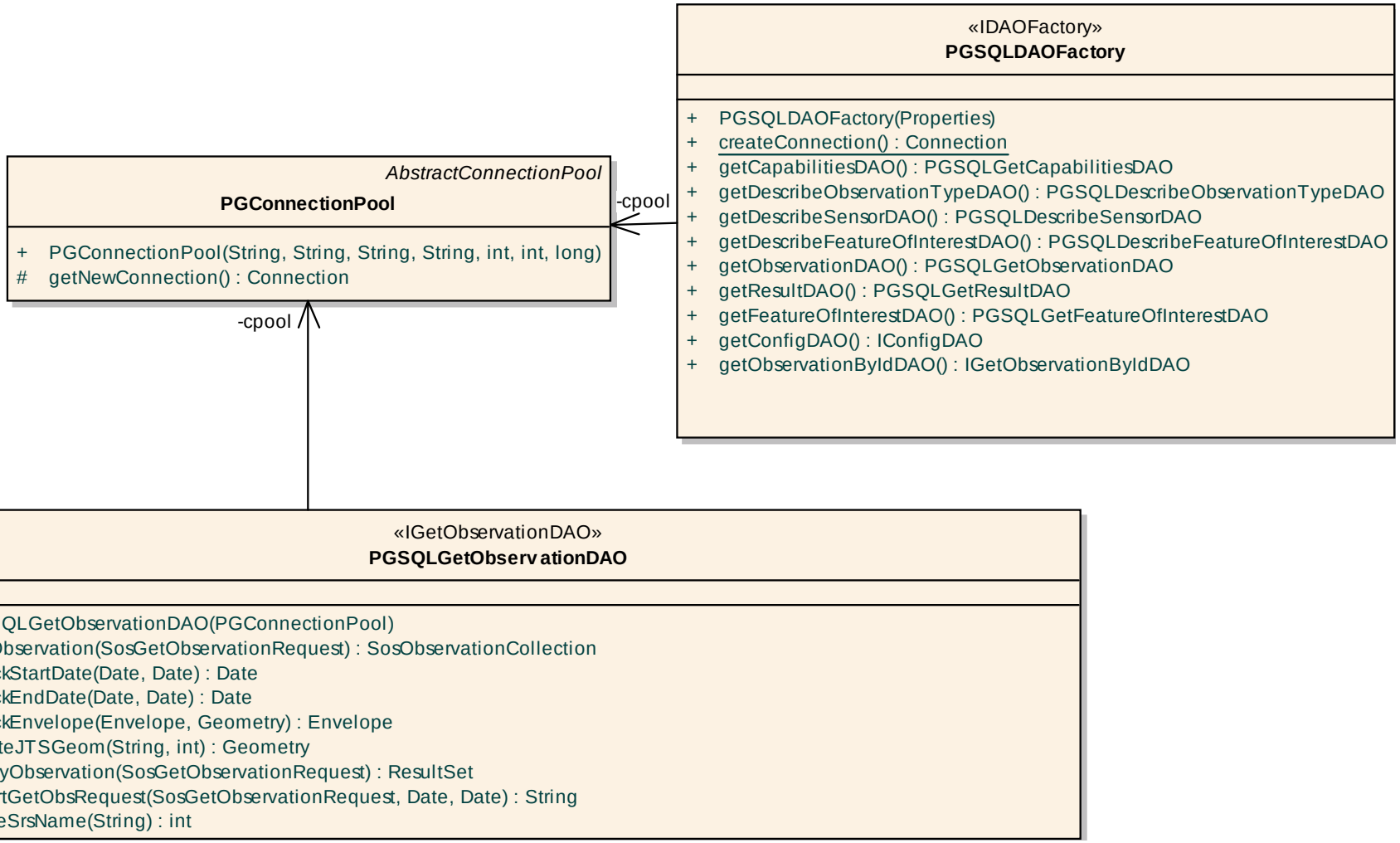

«interface»
IDAOFactory
+ getCapabilitiesDAO() : IGetCapabilitiesDAO
+ getDescribeObservationTypeDAO() : IDescribeObservationTypeDAO
+ getDescribeSensorDAO() : IDescribeSensorDAO
+ getDescribeFeatureOfInterestDAO() : IDescribeFeatureOfInterestDAO
+ getObservationDAO() : IGetObservationDAO
+ getObservationByIdDAO() : IGetObservationByIdDAO
+ getResultDAO() : IGetResultDAO
+ getFeatureOfInterestDAO() : IGetFeatureOfInterestDAO
+ getConfigDAO() : IConfigDAO
«interface»
IGetObservationDAO
+ getObservation(SosGetObservationRequest) : SosObservationCollection


org.n52.sos.ds.pgsql
+ PGConnectionPool
+ PGSQLDAOFactory
+ PGSQLGetObservationDAO

Data Layer

- implementation of `org.n52.sos.ds.IDAOFactory` set in `sos.config` file
- `IDAOFactory` loaded and stored in `org.n52.sos.SosConfigurator`
- `AbstractConnectionPool` for getting connection to DB
- DAO implementation for PostgreSQL/PostGIS in package `org.n52.sos.ds.pgsql`

Data Layer - PostgreSQL



Database

Database schema

- folder db of SOS project contains
  - `datamodel.sql` for creating table structure
  - `test.sql` for inserting example data
  - picture of datamodel (`datamodel.jpg`)
- table for each basic component of O&M model
  - procedure
  - phenomenon
  - feature_of_interest
  - observation
  - quality (following ISO:19113/19114/19115)
  - offering

Extending the service

52north exploring horizons

Implementation of new operations

- Add a new Listener to package `org.n52.sos`, set OPERATION_NAME attribute value to name of implemented operation
- Add listener class name in config file
- Add request class in `org.n52.sos.request` and response class in package `org.n52.sos.resp`
- (Add parsing method in `org.n52.sos.decoder.HttpPostRequestDecoder`)
- Edit `doGetOperation`/`doPostOperation` in `RequestOperator`
- Implement `receiveRequest` Operation in listener
- (Add new DAO for accessing data from database or other datasource)

52north exploring horizons

Implementation of new DAOs

- Create subpackage in `org.n52.sos.ds`
- Create ConnectionPool, which extends `org.n52.sos.ds.AbstractConnectionPool` and implement abstract methods
- Create DAOFactory, which implements `org.n52.sos.ds.IDAOFactory`
- Implement DAO interfaces from `org.n52.sos.ds`

52north exploring horizons

Implemention of further observation types

- Create new class, which represents new observation type, in package `org.n52.sos.ogc.om`
- Class has to extend `org.n52.sos.ogc.om.AbstractSosObservation`
- Add new method in `org.n52.sos.encode.OMEncoder` for encoding new observation class with XmlBeans
- Add creation of new observation objects from ResultSet in `getObservation` method of `org.n52.sos.ds.pgsql.PGSQLGetObservationDAO`

Implemention of further feature types

- Create application schema (or use one of samplingFeatures.xsd schema)
- Compile schema with XmlBeans and add resulting jar to build path
- Create class, which represents new feature, in package `org.n52.sos.ogc.om.sampleFeatures`
- New class must extend abstract class `org.n52.sos.ogc.om.sampleFeatures.SosAbstractFeature`
- Add new method for encoding feature with XmlBeans in `org.n52.sos.encode.OMEncoder`
- Edit implementation of `getObservation()` method in `PGSQLGetObservationDAO` and `getFeature()` in `PGSQLGetFeatureOfInterestDAO`

52north exploring horizons

Installation

Installation (1/2)

- Software needed:
  - PostgreSQL with PostGIS extension (> version 8.1)
  - Apache Ant (> version 1.6.2)
  - Apache Tomcat (> version 5.5)
  - CVS client (e.g. TortoiseCVS)
  - Java JDK/JRE (> version 5.0)
- CVS checkout:
  - `cvs -d:pserver:anonymous@core-52n.cvs.sourceforge.net:/cvsroot/core-52n login`
  - Empty password
  - Modul SOS
- Detailed Installation Guide contained in folder doc (`howToInstall_sos.pdf`)

Installation (2/2)

- Steps:
  - (Install PostgreSQL with PostGIS)
  - Create database and execute file `datamodel.sql` in db folder with PGAdmin
  - Configure properties
    - `Build.properties` file in `conf` folder
    - `Sample_config.properties` file renamed into `config.properties` in `conf` folder
  - Adjust `capabilities_skeleton.xml` (contained in `web/WEB-INF/conf/capabilities`)
  - Create sensor descriptions (remember path for inserting the path into the DB)
  - Populate database (example `test.sql` in `db` folder or through 52north SosFeeder Framework)
  - Build and deploy web application with ant build file `build.xml` in `conf` folder

That‘s it!

- Further information available at:

  https://incubator.52north.org/twiki/bin/view/Sensornet/SensorObservationService

  or ask directly at: swe@52north.org