DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

800 ANNI 1222·2022

UNIVERSITÀ DEGLI STUDI DI PADOVA

SIGNET

signet.dei.unipd.it

WOSS – World Ocean Simulation System – c++ classes

Padova (Italy)

30th of January – 3rd of February 2023

Federico Guerra, Filippo Campagnaro, Michele Zorzi

woss@guerra-tlc.com, filippo.campagnaro@unipd.it

WOSS – the foundation classes

- **`woss::Pdouble`** class that stores a double value and its precision which are used for arithmetic and comparison computations.
  - It is extensively used throughout the whole framework and its containers.
  - By lowering the precision of specific key components (SSP, databases containers etc) it is possible to dynamically perform a sub sampling of the data in use hence simplifying the complexity of the simulation.
- **`woss::Pressure`** class stores an acoustic pressure value for attenuation purposes represented by `complex<double>`. It also provides full arithmetic capability and an attenuation coherency check.
- **`woss::TimeArr`** class represents the channel power delay profile. Its associates a time delay value (`double`) to a `woss::Pressure` value. is It also provides APIs for arithmetic and pressure manipulations.

WOSS – the foundation classes

- **`woss::Coord`** class that represents a geographical coordinate as a pair of **latitude** and **longitude**. It has many APIs in order to compute bearing, distance, etc…
- **`woss::CoordZ`** class that extends **`woss::Coord`** and that represents a geographical coordinate as a tuplet of:
  - **latitude**
  - **longitude**
  - **depth (>= 0 under the sea)**

  It has all the required APIs in order to compute bearing, distance(s), conversion to/from cartesian coordinates, spherical coordinates etc…

WOSS – the foundation classes

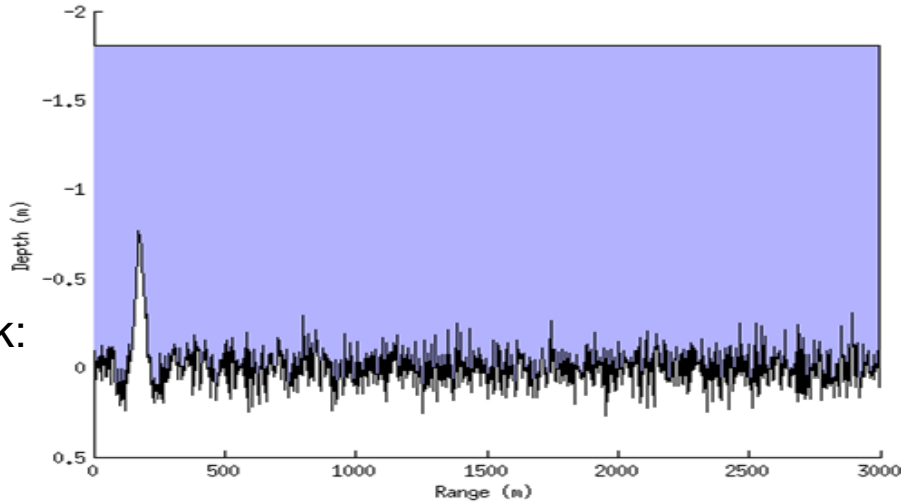
- **`woss::SSP`** class that stores information related to a sound speed profile.
  - For any given depth (**`woss::Pdouble`**) it can associate a
    - temperature [C°]
    - pressure [bar]
    - salinity [psu]
    - sound speed [m/s].
  - It offers capabilities for:
    - arithmetic computations
    - sound speed calculations (Chen and Millero, TEO-10 etc),
    - depth to pressure conversions (and vice versa) with coordinates corrections
    - sound speed profile transformations:
      - Truncation, sub sampling, interpolation
      - Depth extension at constant temperature and salinity
      - random perturbation, the latter is used by the framework when multiple run for each channel simulation are configured.

WOSS – the foundation classes

- **`woss::Sediment`** class provides an interface for creating and manipulating surficial geoacoustic parameters as:
  - compressional wave velocity [m/s]
  - shear wave velocity [m/s]
  - sediment density [g/cm^3]
  - compressional wave attenuation [dB/wavelength]
  - shear wave attenuation [dB/wavelength]

WOSS – the foundation classes

- **`woss::Altimetry`** class implements sea surface wave spectra. Currently there are two models:
  - flat model, which represents a flat sea surface
  - **`woss::AltimBretschneider`** class that implements the Bretschneider (ITTC two parameters) wave spectrum*.
    - Its input parameters are:
      - **H**, characteristic wave height
      - **T**, the wave period [s]
    - Additional parameters which are automatically set by the framework:
      - range [m]
      - range steps
      - scenario depth [m]

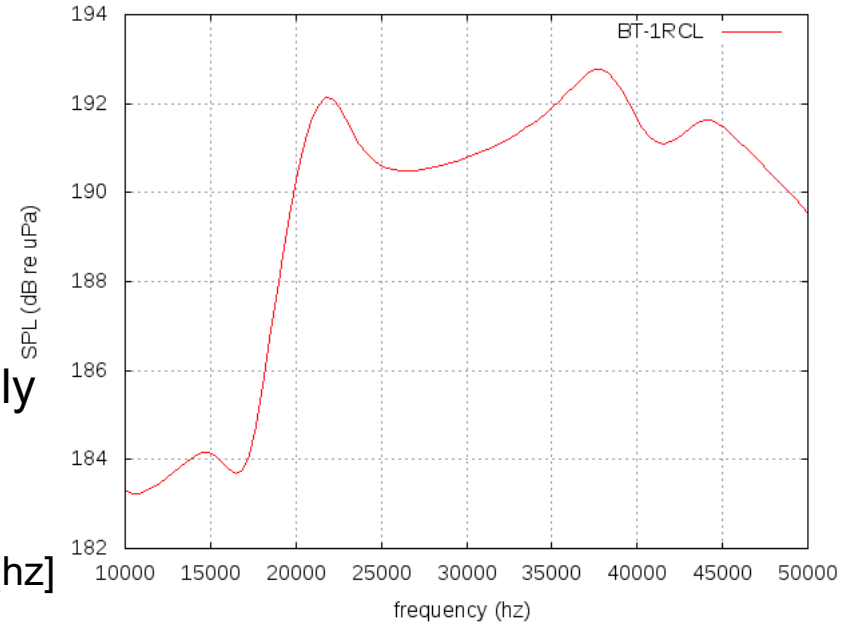


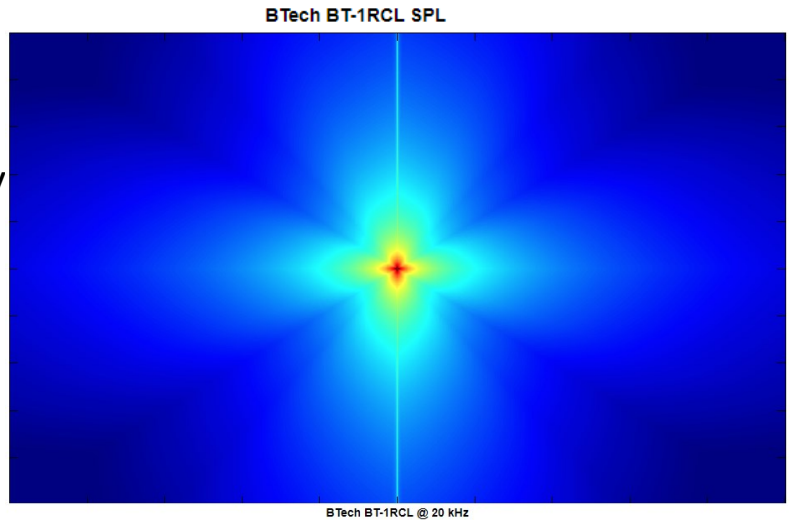
WOSS – the foundation classes

- **`woss::TimeReference`** class that allows WOSS to import a simulator time reference from an external source, usually a network simulator.
- **`woss::Time`** class that provides date representation and its arithmetic APIs.
- **`woss::RandomGenerator`** class that allows WOSS to import random number generator model from an external source (network simulator or else).
- **`woss::Location`** base class that provides APIs for a mobility model based on geographical coordinates (**`woss::CoordZ`**).

WOSS – the foundation classes

- **`woss::Transducer`** class that allows accurate calculation of:
  - power consumption
  - Sound Pressure Level (SPL) output
  - vertical beam pattern
- The model is based on the parameters usually provided by constructor data sheets as:
  - resonance frequency [hz]
  - 3dB bandwith around the resonance frequency [hz]
  - max input power [W]
  - duty cycle [0-1]
  - conductance [uS]
  - Transmitting Voltage Response (TVR) [dB re uPa/V @ 1m]
  - Open Circuit Voltage (OCV) [dB re 1V/uPa]
  - Vertical and horizontal beam pattern





WOSS – the foundation classes

- Transducers can be imported into simulation, and their beam pattern can be rotated or modified with an additive and a multiplicative constant to better suit the user needs.
- **`woss::Location`** supports simulation of dynamic orientation of the associated transducer to better simulate how AUV movements impact on transmission.
- For more information on the **`woss::Transducer`** and current model list please refer to http://telecom.dei.unipd.it/ns/woss/doxygen/transducer_page.html

WOSS – the foundation classes

- **`woss::TransducerHandler`** class that
  - imports **`woss::Transducer`** models from ASCII file
  - associates them to a string into its internal database.
  - provides access of the imported models to the whole framework.

WOSS – the foundation classes

- **`woss::DefinitionsHandler`** singleton class is configured at run-time. It instantiates every foundation class objects for the whole framework.
- The user therefore can define its own classes derived from:
  - **`woss::SSP`**
  - **`woss::Sediment`**
  - **`woss::Pressure`**
  - **`woss::Altimetry`**
  - **`woss::Transducer`**
  - **`woss::TimeRef`**
  - **`woss::RandomGenerator`**

  and plug them in at configuration phase, before the simulation begins.

WOSS – the database classes

- Each database is a **`woss::WossDb`** object. This class separates the database access technology (NetCDF, ASCII, CSV, etc) from the actual operation on the data, thus allowing maximum code reuse.
- Each **`woss::WossDb`** object is instantiated by **`woss::WossDbCreator`** object. The latter has the task of properly create and initialize the returned instance.
- The **`woss::WossDbManager`** has the task of centralizing the access to all the involved databases:
  - Bathymetry database – **`woss::WossBathymetryDb`**
  - SSP database – **`woss::WossSSPDb`**
  - Sediment database – **`woss::WossSedimentDb`**
  - **`woss::TimeArr`** simulation results (optional) – **`woss::WossResTimeArrDb`**
  - **`woss::Pressure`** simulation results (optional) – **`woss::WossResPressDb`**

WOSS – the database classes

- The **`woss::WossDbManager`** has APIs for configuring run-time made environmental databases.
- Each environmental data (bathymetry, altimetry, SSP, sediment) has a set of APIs for setting, importing from file, retrieving and erasing a custom value.
- Each custom data is stored into a RAM container, that has the following key tuple:
  - tx coordinates: the data can be valid for a specific tx coordinate or for all of them if a special value is given.
  - Once the first key is set, the bearing is then considered. Again, a special value can be given in order to be valid for all bearings.
  - Range from the tx coordinates. Also here a special value exists that will consider the data valid for all ranges.

WOSS – the database classes

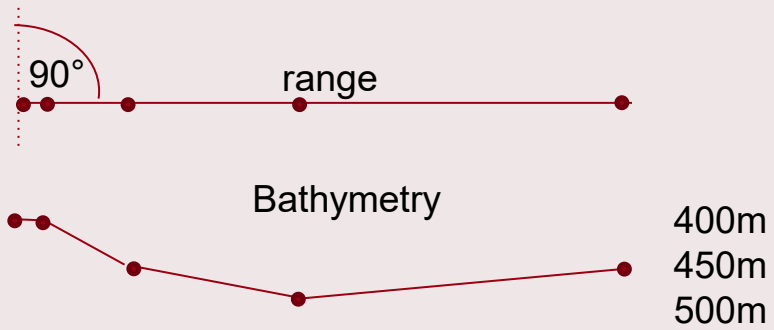
- The getter APIs will then be used by the objects that run the channel simulator.
- The APIs will:
  - search into the custom databases, by getting the data that is closer to the requested coordinates.
  - If the custom databases are not present, then the data will be retrieved from the ordinary databases

WOSS – the database classes

- Custom data databases examples

```
bool setCustomBathymetry( Bathymetry* const bathymetry,
        const Coord& tx_coord = CCBathymetry::DB_CDATA_ALL_OUTER_KEYS,
        double bearing CCBathymetry::DB_CDATA_ALL_MEDIUM_KEYS,
        double range = CCBathymetry::DB_CDATA_ALL_INNER_KEYS );
```

`woss_db_manager->` `setCustomBathymetry(90);`	The key tuple is not given, hence it will take the default special values for all keys. Meaning that all simulations will have a flat bathymetry of 90 meters;

`woss_db_manager->`
`setCustomBathymetry(400, tx, pi/2.0, 1);`
`woss_db_manager->`
`setCustomBathymetry(450, tx, pi/2.0, 10);`
`woss_db_manager->`
`setCustomBathymetry(500, tx, pi/2.0, 40);`
`woss_db_manager->`
`setCustomBathymetry(450, tx, pi/2.0, 100);`
	We have drawed a straight bathymetry line with four points (linearly interpolated): Starting from tx coordinates, bearing 90°
	 

WOSS – the databases

- **GEBCO** NetCDF database is distributed by GEBCO. All data formats up to 2022 are supported in WOSS.
- The **WOA** SSP NetCDF databases contain monthly, seasonal and annual average of sound speed profiles, calculated with the TEOS-10 exact formula, and based on the depth, temperature and salinity provided by the World Ocean Atlas. The available resolution is 1° x 1°. Data are provided at *standard depths* [m]
  - **0, 10, 20, 30, 50, 75, 100, 125, 150, 200, 250, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1750, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500**.
  - The NetCDF files have been generated with NetCDF or NetCDF4 library.
  - Each NetCDF file has the following dimensions:
    - *latitude*
    - *longitude*
    - *Depth*
  - Each NetCDF file has the following variables description:
    - *float ssp(latitude, longitude, depth)* ; //ssp:units = "m/s" ;
    - *float latitude(latitude)* ; // latitude:units = "decimal degrees" ;
    - *float longitude(longitude)* // longitude:units = "decimal degrees" ;
    - *short depth(depth)* ; // depth:units = "m" ;

WOSS – the databases DECK41

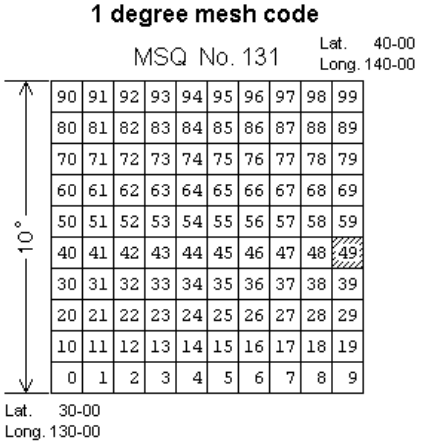
- The **DECK41** sediment database contains Sediment data taken from the NOAA's DECK41 data set.
- The **DECK41** collection contains surficial sediment descriptions for over 36,000 seafloor samples worldwide. Data include collecting source, ship, cruise, sample id, latitude/longitude, date of collection, water depth, sampling device, dominant lithology, secondary lithology, and a brief description of the surficial sediment at the location.

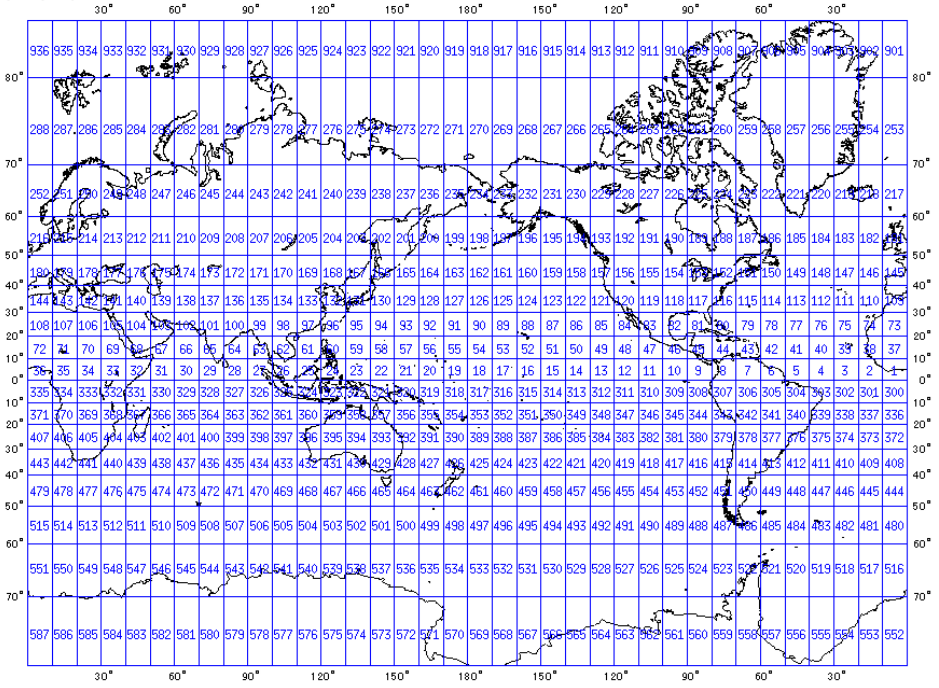
WOSS – the databases DECK41

- The NetCDF files index the dominant and secondary lithology for geoacoustical purposes. These values are represented by an unsigned integer number between 0 and 11:
  1. Coarser than sand
  2. Sand
  3. Silt
  4. Clay
  5. Ooze
  6. Mud
  7. Rocks, Rocks fragment
  8. Organic material (shell, peat, wood, coral, etc.)
  9. Nodules, slab or concretions (manganese, phosphate, iron, glauconite)
  10. Hard bottom
  11. NO_VALUE

WOSS – the databases DECK41

- The data has been organized in three NetCDF databases:
  - The first with the exact coordinates of the data set.
  - A second one with data averaged oven marsden coordinates: marsden square + marsden one degree square.
  - A third one with data averaged over a whole marsden square.





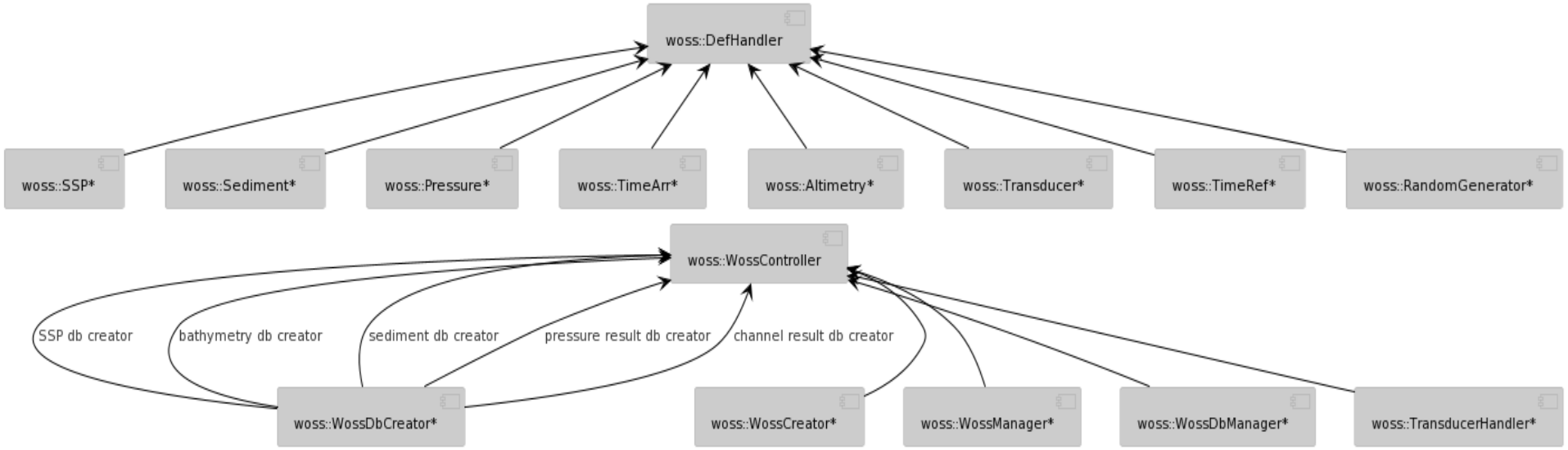
WOSS – the main classes

- The **`woss::Woss`** class provides interface for:
  - configuring, running and reading results of any channel simulator.
  - Averaging and output results over any number of run.
- Each **`woss::Woss`** object is instantiated and configured via a **`woss::WossCreator`** object
- The **`woss::WossManager`** class is the entrypoint of the framework, and offers:
  - The ability of carefully creating and use **`woss::Woss`** instances in a multi-threaded fashion in order to minimize CPU load
  - The ability of planning a strategy for time-varying and/or multi-frequency channel simulations;
  - The ability of storing and retrieving results from a result database, thus saving CPU time when possibile.

WOSS – the main classes

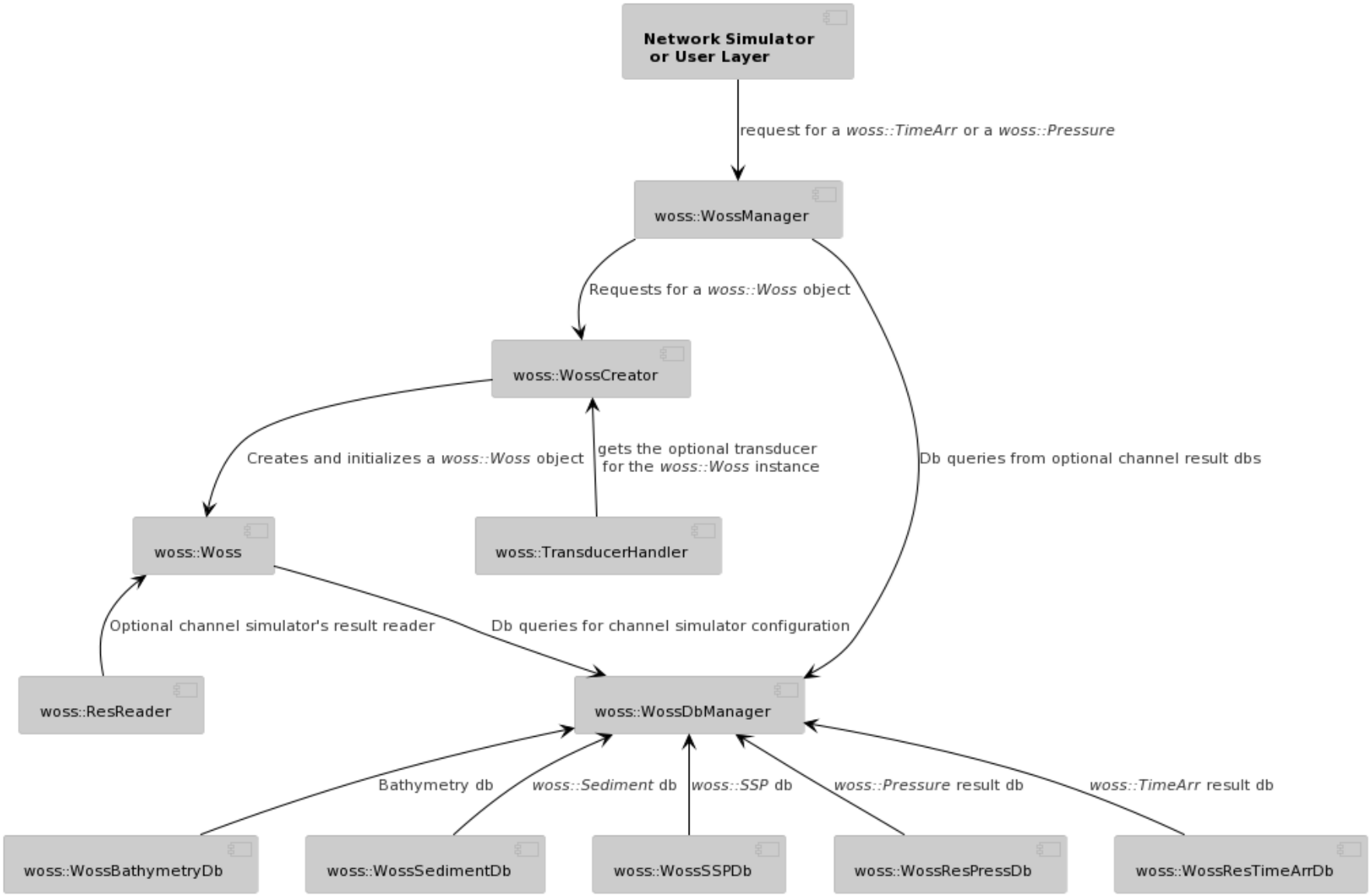
- The **`woss::WossController`** class centralizes and manages all the connections between all the woss objects.
- It needs to be configured with pointers to all the main objects, namely:
  - (optional) each environmental **`woss::WossDbCreator*`**
  - **`woss::WossCreator*`**
  - **`woss::WossManager*`**
  - **`woss::WossDbManager*`**
  - **`woss::TransducerHandler*`**

WOSS – a functional overview



- The init phase can be summarized with the following steps:
  - The singleton **`woss::DefHandler`** must be configured with all the founding class object prototypes
  - The **`woss::WossController`** should be instantiated
  - Each optional **`woss::WossDbCreator`** should be instantiated, configured and plugged into the **`woss::WossController`**
  - The mandatory **`woss::WossCreator`**, **`woss::WossDbManager`** and **`woss::WossManager`** should be instantiated, configured and plugged into the **`woss::WossController`**
  - **`woss::WossController::initialize()`** function should be called

WOSS – a functional overview



WOSS – time evolution

Time evolution is supported by the following classes:

- **`woss::Woss`** - controls its evolution by querying a new **`woss::SSP`** for the new time evolution quantum and by asking its **`woss::Altimetry`** to evolve.
- **`woss::Altimetry`** - performs an evolution of its mathematical model.
- **`woss::AltimBretschneider`** - generates a new realization of its statistical process.

Each class supports an independent time evolution quantum measured in seconds and feature on/off flag

WOSS – time evolution

The framework controls the time evolution through:

- **`woss::WossManager`** – supports queries for a specific **`woss::Time`** object or for a specific number of seconds after the start of the simulation (**`woss::SimTime`**)
- **`woss::WossCreator`** – for each pair of tx, rx **`woss::Location`** the user can configure a specific **`woss::SimTime`** object
- **`woss::WossDbManager`**, **`woss::WossDb`** – all dbs support queries for a specific **`woss::Time`**
- **`woss::WossDbManager`** – A template for custom db data can be used for time dependent custom SSP data. The returned **`woss::SSP`** is a linear interpolation of two SSPs found at the closest **`woss::Time`** values.

WOSS – ns-miracle and DESERT integration

The optional **`woss_phy`** shared library is specific for ns-miracle and DESERT integration and it introduces:

- a new packet header **`struct hdr_woss`**
- **`WossChannelModule`**, **`WossMPropagation`** and **`WossMInterferenceMIV`** classes for channel computations and interference calculations
- propagation effects are considered constant over the bandwidth and the duration of the packet;
- the channel simulator is run at the geometric mean frequency.

WOSS – ns-miracle and DESERT integration

The library also offers:

- a **`ChannelEstimator`**, **`ChEstimatorPlugIn`**, **`ClMsgChannelEstimation`** for channel estimation purposes used at PHY layer
- A **`WossPosition`** and for waypoint mobility with **`WossWpPosition`**
- It provides the **`WossMPhyBpsk`** class for BPSK modulation with power control based on channel estimations and transducer model.
- It provides TCL classes, bound variables and methods in order to properly write and run TCL scripts

References

- Paolo Casari, Cristiano Tapparello, Federico Guerra, Federico Favaro, Ivano Calabrese, Giovanni Toso, Saiful Azad, Riccardo Masiero, Michele Zorzi, "Open-source Suites for the Underwater Networking Community: WOSS and DESERT Underwater", IEEE Network SI “Open source for networking,” 2014
- Federico Guerra, Paolo Casari, Michele Zorzi, “World Ocean Simulation System (WOSS): a simulation tool for underwater networks with realistic propagation modeling”, WUWNet 2009
- Nicola Baldo, Marco Miozzo, Federico Guerra, Michele Rossi, Michele Zorzi, "Miracle: The Multi-Interface Cross-Layer Extension of ns2”, EURASIP Journal on Wireless Communications and Networking Volume 2010, Article ID 761792, doi:10.1155/2010/761792