

Documentation

Sensor World

-

Sensor Network Simulation
Architecture

Contents

1	Introduction	3
2	Messaging System in Sensor World	4
2.1	JMS Java Messaging Service	4
2.2	MantaRay	4
2.3	Implementation in Sensor World	4
3	Components of the Simulation Architecture	5
3.1	Virtual Communication Layer	5
3.1.1	Dictionary System	5
3.2	Sensor Simulation	6
3.2.1	Sensor Behaviour	6
3.2.2	Sensor Design	6
3.3	Phenomenon Simulation	7
3.3.1	Phenomenon Algorithm	7
3.3.2	Phenomenon Design	7
3.4	Visualization Component	7
3.5	Communication Simulation	8
3.5.1	Communication Algorithm	8
3.5.2	Communication Design	9
4	SWE Adapter for Sensor World	9
4.1	Architecture Overview	9
4.2	Sensor Observation Service Adapter	9
4.2.1	Sensor Observation Service	9
4.2.2	SOS Adapter	9
4.3	Sensor Planning Service Adapter	10
4.3.1	Sensor Planning Service	10
4.3.2	Sensor Planning Service Adapter (SPS Adapter)	10
4.3.3	Sensor Planning Service Client (SPS Client)	11

1 Introduction

This documentation provides some information about the Sensor World Simulation Architecture.

Sensor World is a distributed sensor network simulation environment developed at the Institute for Geoinformatics in Münster. It simulates phenomena detected by sensor networks, the physical communication between sensors and different sensor behaviour. The distribution makes it possible to add further components and to run the simulation on multiple machines. Furthermore the architecture offers the exchangeability of simulation components by using interfaces which define how to integrate different simulation models for phenomena, sensors or communication models. In order to link the different simulation components a communication layer has to be established to allow message transfer. This is done by one important core element, the VirtualCommunicationLayer. Overall the architecture comprises the following 4 components, which compose the simulation environment and are described more precisely below:

- VirtualCommunicationLayer
- SensorSimulation
- PhenomenonSimulation
- CommunicationSimulation

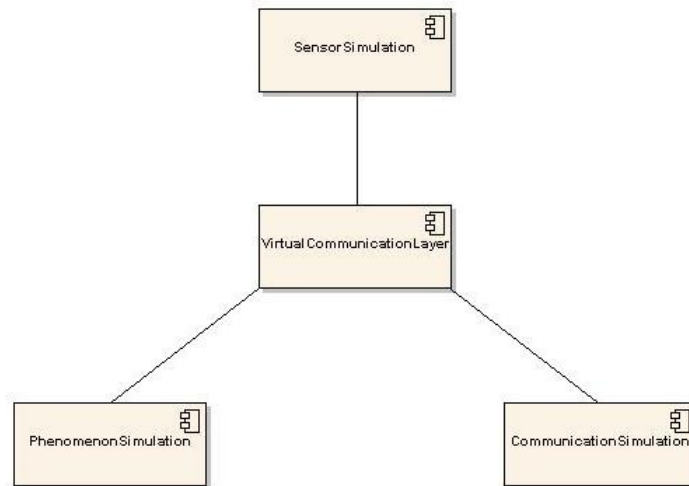


Figure 1: The main components of the sensor network simulation environment.

2 Messaging System in Sensor World

2.1 JMS Java Messaging Service

The JMS API is a messaging standard that allows application components based on the Java 2 Platform, Enterprise Edition (J2EE) to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous. For more information see <http://java.sun.com/products/jms/> for more information, articles etc.

2.2 MantaRay

Mantaray is a distributed, peer-to-peer, server-less communication and messaging solution for JAVA (JMS), C++ and .NET applications. It offers guaranteed delivery, security as well as transactions and supports TCP, SSL and HTTP protocols. (sourceforge.net) In Sensor World MantaRay will be used to connect the different components and is encapsulated by the ICommonConnector class.

2.3 Implementation in Sensor World

Since Mantaray is an implementation of JMS in Sensor World Mantaray classes are only used when creating a Queue or Topic Connection Factory. Afterwards the JMS classes are handling the further connection.

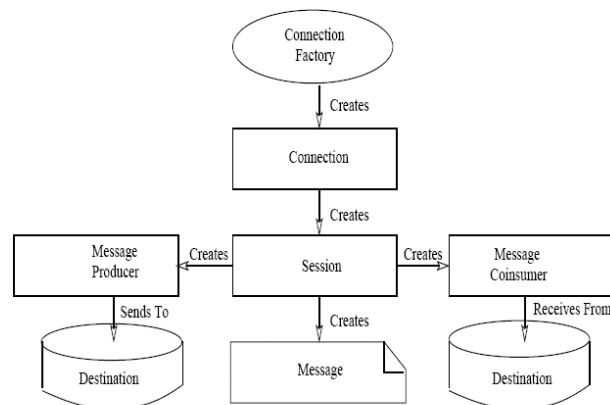


Figure 2: Main elements for the message transfer.

3 Components of the Simulation Architecture

In the following section the main components of the architecture will be introduced and explained in its functionality.

3.1 Virtual Communication Layer

The VirtualCommunicationLayer (VCL) is the connection between all components. "Virtual", because it simulates a communication that will, in reality, be built via many different communication methods and is heavily influenced by the environment of the real sensors and phenomena. Therefore a component is necessary which handles the interaction and communication in the architecture. The messaging system is based on the Java Messaging System JMS with the concrete implementation in MantaRay. The main component of the VCL is the dictionary control class and the dictionary system.

3.1.1 Dictionary System

Every component within the simulation possesses a dictionary to deal with one important task of the virtual communication layer: mapping the messages depending on the message type to the correct destination. More precisely a dictionary contains information about components in the system and their destinations at every time. The dictionary control component forms the centralised communication and takes care of the actuality of all dictionaries and guarantees that all instances have the same knowledge at every time about the sensor network. Regarding the differences between sensor, phenomenon and communication it is necessary to distinguish between 3 types of dictionaries.

- Sensor Dictionary
- Phenomenon Dictionary
- Communication Simulation Dictionary

The functionality for managing and updating these dictionaries to guarantee a consistent view for all components is provided by the DictionaryControl class. It provides methods to publish new dictionaries when a new component registers in the system and accordingly update dictionaries for the others. To react to the different events the dictionary control receives a dictionary message with a specific task for creating, updating and deleting the dictionaries and destinations.

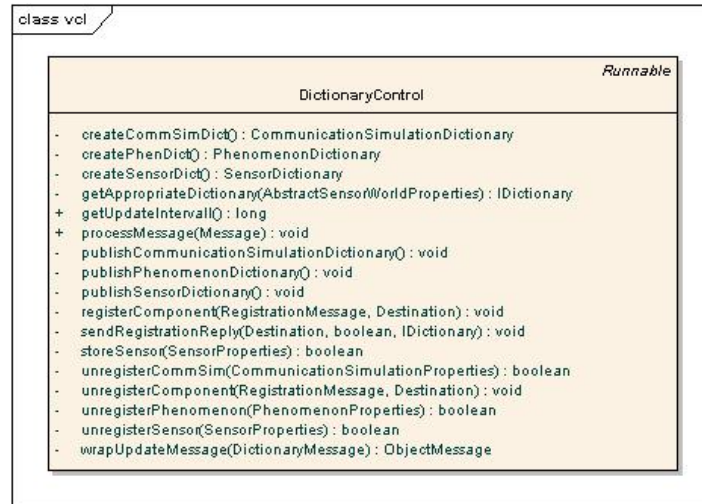


Figure 3: Main elements for the message transfer.

3.2 Sensor Simulation

The sensor simulation component consists of a number of separate independent sensor simulation processes, which emulate the behaviour of a single sensor. The sensor behaviour for example is determined by the manner of the movement like speed, direction, detection strategies and measurement results.

3.2.1 Sensor Behaviour

At the moment the default behaviour of a sensor is based on a random movement in a predefined bounding box with various speed values.

3.2.2 Sensor Design

Each sensor is linked to the virtual communication layer by its specific sensor connector, where the messaging structure is encapsulated. The connector provides a bridge that forwards outgoing messages and accepts incoming messages from the message layer. As the processing of incoming messages is strongly dependent on the sensor type the insertMessage method has to be implemented to define the reaction to the different incoming messages types. In addition the implementation of a sensor needs information about the behaviour of the sensor. More precisely it has to be defined which actions the sensor is performing throughout his lifetime. The living cycle of a sensor is simulated through a run method in which the different aspects of the sensor behaviour (move method) are controlled and conducted in a loop. The move method is responsible for calculating the next sensor position due to the sensorable behaviour factors.

It is easily possible to implement different sensors with a specific behaviour by using the Abstract Sensor Simulation class. This class delivers the functionality to link the sensor to the simulation and offers relevant methods like:

- move()
- doMeasurement()

3.3 Phenomenon Simulation

The phenomenon simulation has the task to simulate a phenomenon the sensor network is working on. To make measurement in the system possible different kinds of phenomenons can be simulated just as dispersion/spreading and provide these values as offerings to be measured by sensors.

3.3.1 Phenomenon Algorithm

The default phenomenon simulation is based on a multi phenomenon composed of multiple single phenomenon layers, which change their sizes and dummy values based on a number of parameters (e.g. speed) in the scope of a bounding box.

3.3.2 Phenomenon Design

The phenomenon simulation provides an interface that allows sensors to execute virtual measurements. To implement a phenomenon simulation the `AbstractPhenomenonSimulation` Class is used which provides connection to the messaging layer and offers the `doMeasurement` method. This method has to be overwritten just like the `getRasterSnapshot` method, important for the `VisualizationComponent`.

The `doMeasurement` method receives a `MeasurementRequestMessage` as a parameter which contains information about the position of the sensor, the geometry to be measured (point/area) and the phenomena the sensor is able to detect. Based on these information the `doMeasurement` method returns the phenomenon value (temperature, concentration,...) at a certain position encapsulated in a `Measurement Response Message`.

3.4 Visualization Component

The `SensorWorldVis` Controller provides the possibility to create a video of the simulation cycle to show the activity of the phenomenon and sensors in `Sensor World`. The controller is defined by a runtime and interval step number and calls a method `drawJpeg` in which the phenomenon raster and sensor position are combined in one layer. For every interval step one image is created and in the end merged in one visualization video.

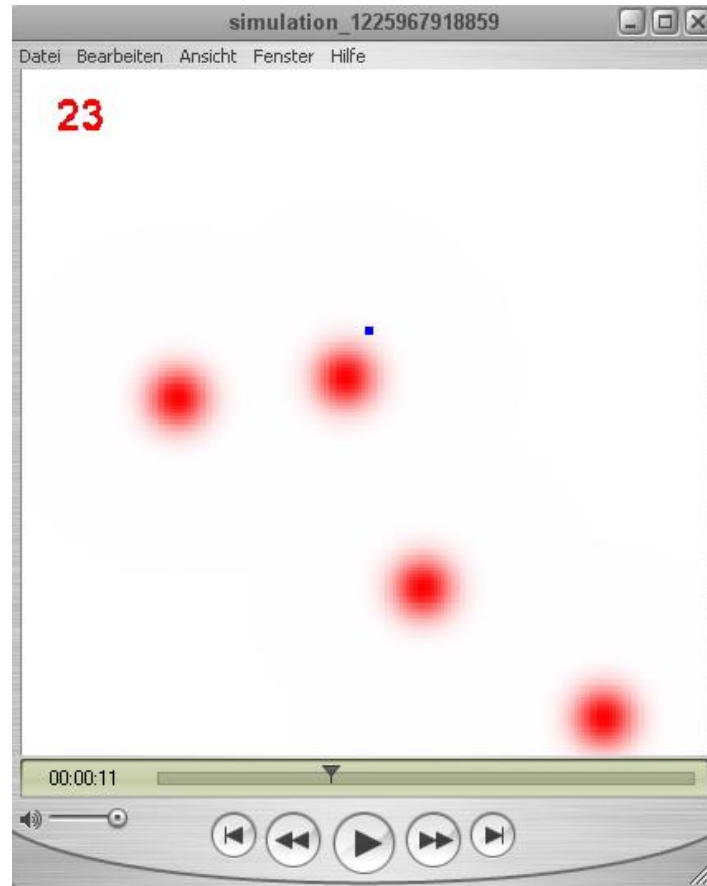


Figure 4: Screenshot of the visualization video.

3.5 Communication Simulation

The Communication Simulation brings "the real world" into the simulation. Since the sensors in the simulation do not communicate through a physical medium it is necessary to determine the connectivity within the network. Thus all communication between sensors is channelled through a Communication Simulation component to check the reachability first.

3.5.1 Communication Algorithm

By default the Communication Simulation is not used and since the simulation is based on the assumption that communication between all components in Sensor World is possible.

3.5.2 Communication Design

The interface of the communication simulation has two important aspects. In the first place the simulation needs information about the structure of the sensor network (input methods) and secondly methods for determining the connectivity are necessary (reachability methods). In order to define a restricted communication the Abstract-CommunicationSimulation class has to be implemented together with the methods to construct a model of the communication network and to ask for the reachability.

4 SWE Adapter for Sensor World

4.1 Architecture Overview

In the following diagram the communication connections between all components of the architecture are illustrated.

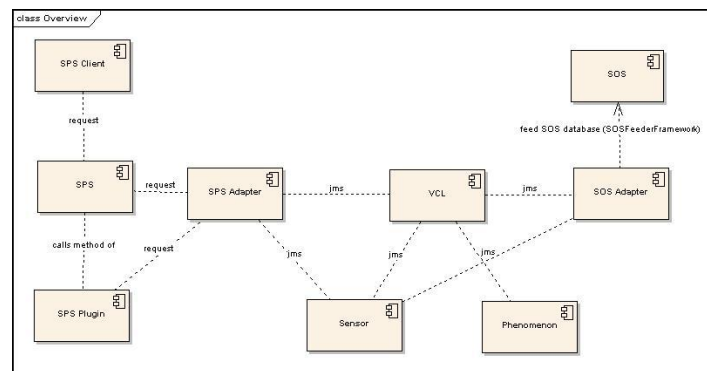


Figure 5: SPS Client User Interface

4.2 Sensor Observation Service Adapter

4.2.1 Sensor Observation Service

”The OGC Sensor Observation Service Interface Standard (SOS) provides an interface for managing deployed sensors and retrieving sensor data and specifically observation data. Whether from in-situ sensors (e.g. water monitoring) or dynamic sensors (e.g. satellite imaging), measurements made from sensor systems contribute most of the geospatial data by volume used in geospatial systems today.”¹

4.2.2 SOS Adapter

Through the connection to the SOS it is possible to make sensors and sensor data archives accessible via an interoperable web based interface. The connection is estab-

¹<http://www.opengeospatial.org/standards/sos>

lished via an adapter, which is running as a separate java application and listens to registration and measurement messages send in the simulation network.

The basic `SosAdapter` class handles the Mantaray connections and creates the consumers for the different topics (communication channels) used for measurement and registration messages. The class processes the incoming messages and passes it to the `Sos Adapter Application` class in which the final insertion to the database is happening. Before that, the database has to be initialized by using the database DAOS, predefined from the `SOSFeeder`, a framework which inserts data in the standard SOS database. When receiving a registration message of a sensor (procedure) or of a phenomenon it will be added to the database and necessary relationships for the SOS data-model will be inserted using the insertion and relationship daos. Afterwards it is possible to feed the database with observations of the sensor and also track its movement by creating a history of the procedures. Through the web interface the feeded data is always callable via requests.

4.3 Sensor Planning Service Adapter

4.3.1 Sensor Planning Service

”The OGC Sensor Planning Service Interface Standard (SPS) defines interfaces for queries that provide information about the capabilities of a sensor and how to task the sensor. The standard is designed to support queries that have the following purposes: to determine the feasibility of a sensor planning request; to submit such a request; to inquire about the status of such a request; to update or cancel such a request; and to request information about other OGC Web services that provide access to the data collected by the requested task.”²

4.3.2 Sensor Planning Service Adapter (SPS Adapter)

The idea of connecting Sensor World with the SPS is to task the sensors via the framework of the SPS. For this purpose its necessary to implement a SPS plugin suitable for the Sensor World architecture. The `Sensor World SPSPlugin` is the connection between the SPS Service and the Sensor World environment. Through the SPS plugin pattern it is possible to register various types of taskable sensors and make them accessible through the SPS framework. The first important step for the configuration is the implementation of the `Sensor Interface` which contains necessary methods for the definition of the sensor behaviour.

- `sps:GetFeasibilityRequest` (optional)
- `sps:SubmitRequest` (mandatory)
- `sps:UpdateRequest` (optional)
- `sps:GetStatusRequest` (optional)

²<http://www.opengeospatial.org/standards/sps>

- sps:CancelRequest (optional)

To make the plugin configurable all important sensor informations are written in the SensorConfiguration document (xml) which is processed by the SPS Plugin. This document is used during the start-up and registration process of a new sensor and stores the information in the SPS database.

4.3.3 Sensor Planning Service Client (SPS Client)

The SPS Client receives the information about the available sensors via the SPS capabilities document and provides a graphical user interface for controlling sensors in Sensor World. On the basis of these capabilities information the client filters necessary parameters for tasking and displays them according to the choice of the user. The user can choose a specific sensor to control and track it via the control and status console panel. Furthermore the input validation for the tasking happens on the client side before the request is send to the SPS.

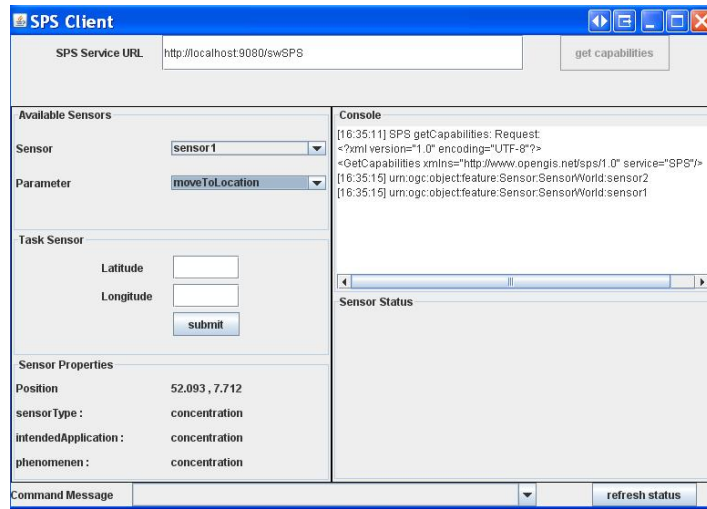


Figure 6: SPS Client User Interface