**UnionVms**

**Developing new Modules**

Joakim Johansson

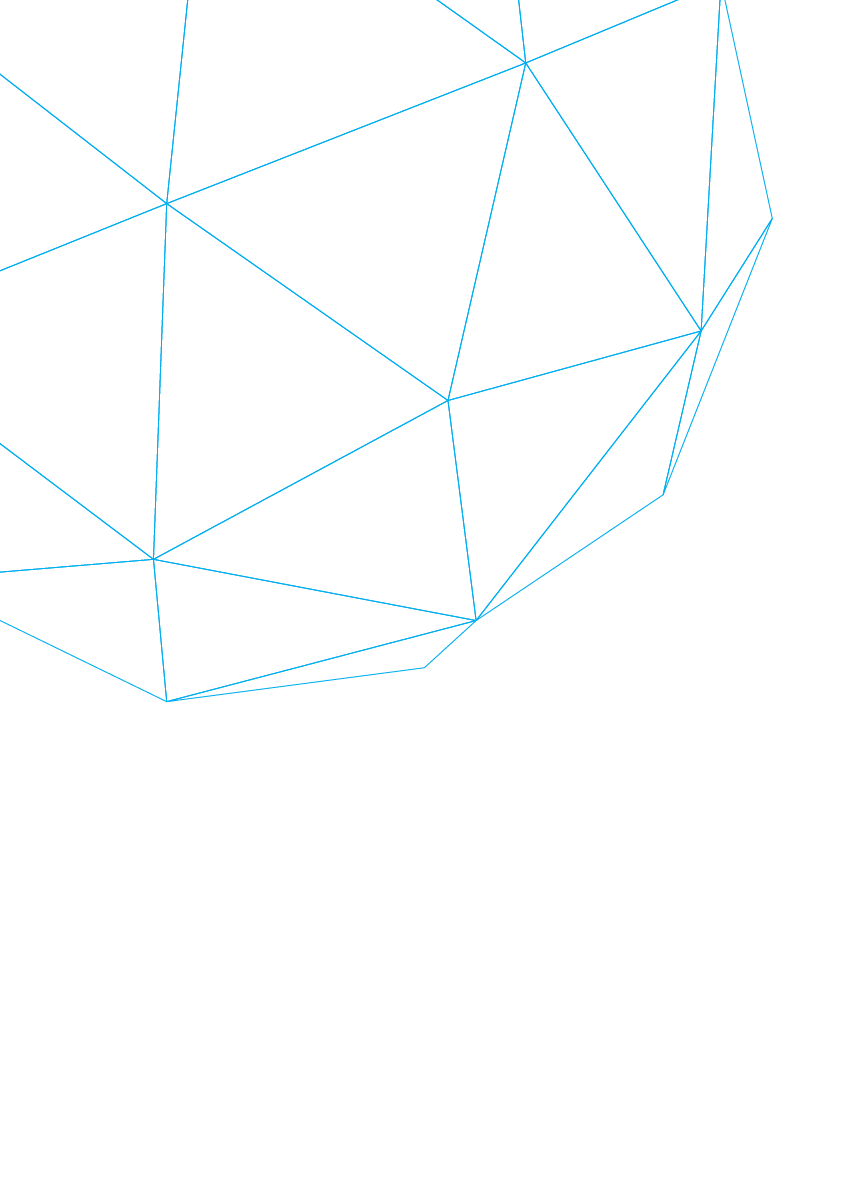


Table of contents

[Document historyDeveloping new modules 3](#_Toc445123749)

[Key koncepts of the main module. 4](#_Toc445123750)

[Model 4](#_Toc445123751)

|  |  |  |
| --- | --- | --- |
| Author | Changes | Date |
| Joakim Johansson | Created document | 2016-03-07 |
|  |  |  |

# Document historyDeveloping new modules

Before you start reading this document make sure you have read the document ”Initial reding.docx” in the ”General documentation” folder. This document descibes a general overview of how a module work, tips on how to get started and commands you can use to ease the development of the module.

If you want a more in depth on how to setup a development environment you should look at the document ”Setup Local Environment Guide.docx” in the folder where this document can be found.

## Key koncepts of the main module.

### Model

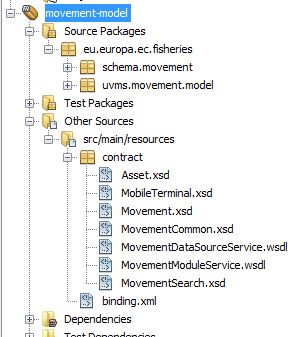
The purpose of the model package is to provide the interface to other modules who wants to make queries, upserts, deletes etc.

Figure Movement modules model structure

To achieve this the module uses a plugin to generate POJOs from a predefined contract. The contract exists of a .wsdl:s and .xsd:s.

The module has 2 interfaces that needs to be defined. This is because the module should use either one or several database module\modules or proxy module\modules. The reason to why theese interfacese are separated is that we need a separation between what is possible to do from the JMS queue and the database queue. We need to concider that we will have a third way of communicating in the module namely the REST api. We might want to have certain queries done for the rest api that we do not want to expose through the JMS queue. Still we will need to make the specific queries to the database layer.

\*DataSourceService.wsdl

This interface defines the communication between the main module and the proxy\database module. This interface can be seen as a combination to serve the JMS queue and the REST api of the module.

#### \*ModuleService.wsdl

This interface defines the interface that the module exposes to other modules on the JMS queue. This should NOT be confused with the DataSourceService.

#### xsd

It is strongly adviced to try to modularize the model in to understandable pieces. For example in the movement this modularization concists of several xsd:s. one xsd should represent one domain for the model.

### Dependencies in the model

The model package cannot have dependencies to other modules models. The reasons to that is as follows. If we have 3 modules that want to interact via the JMS queue they all have their models that define their interfaces.

The import of the dependency from Module A to Module B is defined in the pom.xml of the service layer in the module. All imports in this module will affect the whole module because all logic for the module is done in the service layer. As illustrated below.

If Module A wants to communicate with Module B, Module A imports Module Bs Model, in our case Version 1.

Module C then wants to communicate with Module A. Module C imports Module As Model Version 1. Module C now has dependencies to Module A Version 1 and Module B Version 1 by indirect depencies in Maven.

In our case module C also wants to communicate with Module B. But the version for Module Bs model has changed but it is only Module B that wants that version and not Module A. In this case we cannot know which version that will be available for Module C at compile time. During the project we had theese problems and the sollution was that no dependencies were allowed in the model layer.

Figure Module dependencies, how NOT to do

Module A

Model Version 1

Module C

Module B

Model Version 1

Model Version 1

Model Version 2

Module B

The rules for dependncies applies for all dependencies in the model layer. If you use any type of common library or helper library that will propagate to the module that imports the model and may affect the model in a bad way.

So as a rule you should NOT add more dependencies in the model layer than there already are. j

### Configure the POM to generate sources

The only thing you need to do to get up and running with the model project is to rename the two .wsdl files to the names you want. Then update the properties in the POM file in the model folder. The two properties you havet o change are:

\\hav.havochvatten.se\hav\root\users\jojoha\Desktop\FOLDERS\Pictures\dependencies.JPG

Figure Properties from movement model POM