Deliverable D4.1.2 Concerted Approach V2
Appendix 2: Back-back end documentation: hydrology

Revision history			
Version	Date	Modified by	Comments
0.1	2011-08-16	Initial version	Initial version – back-back end service documentation
0.2	2011-08-22	Esa Falkenroth	Simplified API-spec
0.3	2011-08-25	Esa Falkenroth	Even more simple API-spec
0.4	2011-08-25	Esa F	Adjusted names of methods to match those of the prototype implementation
0.5	2011-08-26	Esa + Tomas	Added upstream. Added description and examples of forcing files and resultfiles.
0.6	2011-09-12	Esa	Changing document to match the example program of Java API.
0.7	2011-09-28	Mihai Bartha/ AIT	Reviewed and completely reformatted the document. Agree to most comments provided by Martin Scholl/CISMET. Commented the document
0.8	2011-10-03	Jens + Claes	Revised document according to feedback. Current changes of the system are reflected.

Copyright © SUDPLAN

Table of Content

TABLE OF CONTENT **2**

1. COMMON SERVICES – BACK-BACK END HYD **3**

1.1 OVERVIEW OF API USAGE 4
1.2 TIME SERIES RESULTS 4
1.3 UC-831 VISUALIZE HYDROLOGICAL INFORMATION ON THE PAN EUROPEAN SCALE 4
1.4 UC-832 AUTO CALIBRATION OF CS HYDROLOGICAL MODEL 4
1.5 UC-833 EXECUTE CS HYDROLOGICAL MODEL 5
1.6 DETAILED API DESCRIPTION 5
1.7 DESCRIPTION OF TERMS 5
1.8 SUDPLANHYPEAPI CONSTRUCTOR 6
1.8.1 runSimulation 6
1.8.2 createSimulation 6
1.8.3 setPointOfInterest 6
1.8.4 setSimulationTime 6
1.8.5 setDefaultScenario 7
1.8.6 getTimeSeries 7
1.8.7 storeTimeSeries 7
1.8.8 listTimeSeries 8
1.8.9 describeTimeSeries 8
1.8.10 getExecutionStatus 9
1.8.11 getResultfileStatus 9
1.8.12 storeSimulationResult 9
1.8.13 listScenarios 10
1.8.14 createSimulation 10
1.8.15 createCalibrationSimulation 10
1.8.16 createSubmodel 10
1.8.17 useCalibrationFrom 11
1.8.18 mergeObservations 11
1.8.19 deleteExecution 11
1.9 DATA TYPES 11
1.9.1 Sample 11
1.9.2 ExecutionStatus 12
1.9.3 Scenario 12
1.9.4 TimeSeriesMetadata 12
1.10 WEB FEATURE SERVICE 13

1. Common Services – Back-back end HYD

This document describes the back-back end of the hydrology part of the common services, i.e. web-services to run Hype-simulations and store the results of these model runs. Hype is a hydrological model. An overview of the architecture is given in Figure 1.

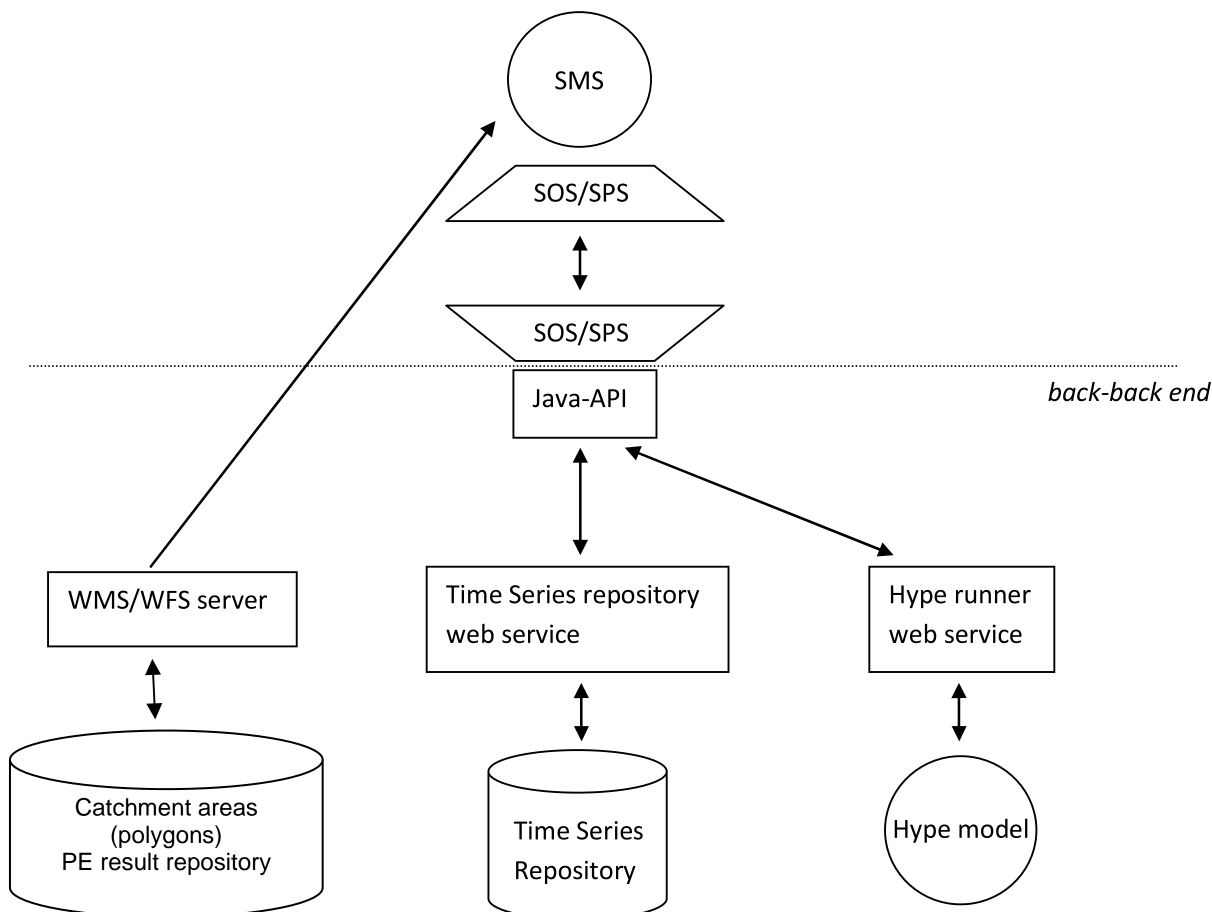

Figure 1. General architecture for the CS hydrological downscaling.

The back-back end of the common services consists of:

- Web-service to run Hype simulations
- Web-service to store time series, specifically results from Hype simulations
- Java-API to access those web services
- WMS/WFS server for access to geographic data

This document describes the java-API and how it can be used by the SOS/T-SPS back end to run simulations and handle time series. It specifically describes what operations to perform when implementing the use cases:

- UC-831 “Visualize hydrological information on the pan European scale”
- UC-832 “Auto calibration of CS hydrological model”

- UC-833 “Execute CS hydrological model”

The back-back end consists of a model runner and a time series repository. Therefore there are methods to run simulations and methods to store and retrieve simulation results in the repository. Looking at the API it is useful to have this distinction in mind.
A WMS/WFS server is also part of the back-back end to service map layers and geographic information on basins that can be used for example to visualize upstream basins.

1.1 Overview of API usage

One of the first basic operations is to list available Scenarios. A Scenario can be for example “NORMAL” or “ECHAM5”. A Scenario is used as a base for a simulation.
Using one of the available Scenarios, a Simulation can be created. Creating a Simulation creates a work area at the “Model runner”. A handle to this work area is returned, that can be used to set properties and perform calibration on the simulation before execution. Operations that can be performed can be setting time of the simulation and setting point of interest.
When properties are set, a simulation can be started. As a result a handle to the running execution is returned. This handle can be used to poll the server of status of the running process. It is also possible to get the current progress of the created result using the simulation handle that started the execution.
When the execution status indicates that the execution is done the result can be stored in the “Time series repository”. When storing the simulation result a unique identifier is connected to the time series for later retrieval.
Time series can be stored in and retrieved from the “Time Series Repository”. The repository can also be queried to retrieve metadata about time series or to get all time series identifiers for a specific basin.

1.2 Time series results

The result when retrieving a time series is simply a list of date-value pairs. To get information about the specific time series there is a separate operation that describes the time series. This operation returns information such as unit, resolution and which variable the time series represents.

1.3 UC-831 Visualize hydrological information on the pan European scale

This section describes necessary steps used in this use case.

- See available Scenarios: listScenarios.
- Retrieve daily time series: getTimeSeries.

1.4 UC-832 Auto calibration of CS hydrological model

This section describes necessary steps used in this use case.

- Upload observed time series: storeTimeSeries

Copyright © SUDPLAN

- Prepare a calibration
  Calibrate Model: createCalibrationSimulation
  Retrieve a previously stored time series: getTimeSeries
  Add time series to model: mergeObservations
  Create sub model: createSubmodel
  Wait until it is done: getExecutionStatus, until status is DONE.

- Start the calibration simulation: runSimulation
  Wait until it is done: getExecutionStatus, until status is DONE.

- Simulation progress visualization: getResultfileStatus

- Handle simulation result
  Store simulation result in result repository: storeSimulationResult

- Retrieve simulation result: getTimeSeries

1.5 UC-833 Execute CS hydrological model

This section describes necessary steps used in this use case.

- See what Scenarios exists: listScenarios

- Prepare model run
  Create Simulation: createSimulation
  Select simulation period: setSimulationTime
  Select POI: setPointOfInterest

- Start model run: runSimulation
  Wait until it is done: getExecutionStatus, until status is DONE.
  Simulation progress visualization: getResultfileStatus

- Store result in result repository: storeSimulationResult

- Retrieve simulation result: getTimeSeries

1.6 Detailed API description

1.7 Description of terms

1.7.1 subid

The identifier of a subbasin. This is available in a WFS layer and should be provided by the SMS client.

1.7.2 simulationId

A handle to a simulation. A simulation is a HYPE (hydrological model) setup. Use runSimulation to actually start the simulation run.

1.7.3 executionId

A handle to a started execution.

1.8 SudPlanHypeAPI constructor

Create a new api instance

```
public SudPlanHypeAPI(String host)
```

Parameter	Type	Description
host	String	The hostname of the server (and optionally a port), eg "localhost:8080"

1.8.1 runSimulation

Run the simulation.

```
public String runSimulation(String simulationId)
```

Parameter	Type	Description
simulationId	String	The simulation to run

1.8.1.1 Returns

Type	Description
String	A handle to the ongoing execution

1.8.2 createSimulation

Create a new empty simulation to be modified by subsequent actions.

```
public String createSimulation()
```

1.8.2.1 Returns

Type	Description
String	The identifier of the new simulation

1.8.3 setPointOfInterest

Set point of interest, ie the basin whose discharge one is interested in. This will create a minimal simulation configuration.

```
public void setPointOfInterest(String simulationId,
                               int basinId)
```

Parameter	Type	Description
simulationId	String	The id of the simulation to modify
basinId	int	The basin of interest

1.8.4 setSimulationTime

Set date interval for the simulation. Dates are pretty much given by the scenario, but they can be modified.

```
public void setSimulationTime(String simulationId,
                              String startDate,
                              String printDate,
                              String endDate)
```

Parameter	Type	Description
simulationId	String	The id of the simulation to modify
startDate	String	The first date to simulate. Date format: YYYY-MM-DD
printDate	String	The first date to include in the result (the first years of the simulation run is a warm-up period) Date format: YYYY-MM-DD
endDate	String	The last date to simulate. Date format: YYYY-MM-DD

1.8.5 setDefaultScenario

Set scenario to use for a simulation.

```
public void setDefaultScenario(String simulationId,
                               Scenario scenario)
```

Parameter	Type	Description
simulationId	String	The id of the simulation to modify
scenario	Scenario	The scenario to use

1.8.6 getTimeSeries

Retrieve a time series by name, subid and date interval.

```
public List<Sample> getTimeSeries(String timeSeriesId, int subid,
String startDate, String endDate)
```

Parameter	Type	Description
timeSeriesId	String	The name of the time series
subid	int	The subid to get data for
startDate	String	The first date you want data for (if available) Date format: YYYY-MM-DD
endDate	String	The last date you want data for (if available) Date format: YYYY-MM-DD

1.8.6.1 Returns

Type	Description
List<Sample>	A list of samples

1.8.7 storeTimeSeries

Store a timeseries under a given name using provided metadata. Data is read from an input stream using the following file format. The file format is tab separated columns. The rows are scanned from the top until a row whose first column matches DATE (case is not significant). This row is used as the header row and columns 2- are taken to be subids. Each subsequent row should have an ISO date (YYYY-mm-dd) in the first column and a double (as a string, eg 3.125) for all the other columns. Time series are assumed to be dense (no gaps between dates) and all rows must have the same number of columns. If there are gaps they must have the value -9999 to indicate that the value is missing.

```
public void storeTimeSeries(String timeSeriesId,
                            String variable,
                            String unit,
                            String resolution,
                            String startDate,
                            InputStream input)
```

Parameter	Type	Description
timeSeriesId	String	The name of the time series. This must be unique.
variable	String	The name of the variable.
unit	String	The name of the unit.
resolution	String	Time step resolution
startDate	String	The date of the start of the time series. Date format: YYYY-MM-DD
input	InputStream	The data stream to read from.

1.8.7.1 File format

The first row includes a text string (e.g. ‘date’, no white space allowed) and then id for subbasins. The first column is date in the format YYYY-MM-DD. The second to last columns are values for all subbasins. Columns are separated by whitespace (tab or space). No missing values may exist.

Example:

```
date        100001      100002      100003
1996-01-02  3.25        7.8         9.1
1996-01-03  3.45        7.9         9.0
…
```

1.8.8 listTimeSeries

List the available stored time series for a given subid

```
public List<String> listTimeSeries(int subid)
```

Parameter	Type	Description
subId	int	The subid to find time series for.

1.8.8.1 Returns

Type	Description
List<String>	The list of time series identifiers

1.8.9 describeTimeSeries

Copyright © SUDPLAN

Get metadata for a timeseries.

```
public TimeSeriesMetadata describeTimeSeries(String timeSeriesId)
```

Parameter	Type	Description
timeSeriesId	String	The id of the timeseries.

1.8.9.1 Returns

Type	Description
TimeSeriesMetadata	Timeseries metadata

1.8.10 getExecutionStatus

Get the status of a running execution that has been started by either runSimulation or createSubmodel.

```
public ExecutionStatus getExecutionStatus(String executionId)
```

Parameter	Type	Description
executionId	String	The id of the execution.

1.8.10.1 Returns

Type	Description
ExecutionStatus	The process status.

1.8.11 getResultfileStatus

Check how far a simulation has come.

```
public int getResultfileStatus(String simulationId)
```

Parameter	Type	Description
simulationId	String	The simulation id to check.

1.8.11.1 Returns

Type	Description
int	The estimated progress as percentage points. (a value between 0 and 100)

1.8.12 storeSimulationResult

Store the result of a simulation. The method retrieves the simulation result and stores it under a specified name.

```
public void storeSimulationResult(String simulationId,
                                  String startDate,
                                  String timeSeriesId)
```

Parameter	Type	Description
simulationId	String	The id of the simulation whose result should be stored.
startDate	String	The start date of the simulation (would

		most likely be the startdate of the scenario used to create the simulation. Date format: YYYY-MM-DD
timeSeriesId	String	The name under which to store the result. This name must be unique. Existing timeseries won't be replaced.

1.8.13 listScenarios

List available scenarios. Scenarios are used as a baseline for a simulation.

```
public List<Scenario> listScenarios()
```

1.8.13.1 Returns

Type	Description
List<Scenario>	The list of available scenarios.

1.8.14 createSimulation

Create a simulation for a given scenario. The default scenario settings are used.

```
public String createSimulation(Scenario scenario)
```

Parameter	Type	Description
scenario	Scenario	The scenario to base the simulation on.

1.8.14.1 Returns

Type	Description
String	The identifier of the created simulation.

1.8.15 createCalibrationSimulation

Create a calibration simulation for a given subid.

```
public String createCalibrationSimulation(Scenario scenario,
                                          int subid)
```

Parameter	Type	Description
scenario	Scenario	The identifier of the scenario.
subid	int	The subid to calibrate for.

1.8.15.1 Returns

Type	Description
String	The identifier of the new simulation.

1.8.16 createSubmodel

Create the submodel to use in calibration. It is wise to run the calibration for only a part of Europe to cut down the runtime. This starts a long running process on the server side.

```
public String createSubmodel(String simulationId)
```

Parameter	Type	Description
simulationId	String	The identifier of the simulation as returned by createCalibrationSimulation.

1.8.16.1 Returns

Type	Description
String	The id of the execution.

1.8.17 useCalibrationFrom

Reuse the calibration result from a previous calibration.

```
public void useCalibrationFrom(String newSimulationId,
                               String calibrationSimulationId)
```

Parameter	Type	Description
newSimulationId	String	The newly created simulation that you intend to run.
calibrationSimulationId	String	The (already run) calibration simulation id to copy parameters from.

1.8.18 mergeObservations

Merge several series of observations into one and upload it to a calibration simulation. Use this to create input to the calibration run.

```
public void mergeObservations(String calibrationSimulationId,
                              String[] subids,
                              List<List<Sample>> newQobs)
```

Parameter	Type	Description
calibrationSimulationId	String	The simulation to add observations to.
subids	String[]	The subids where the observations should be used. Must be in the same order as the list of sample series.
newQobs	List<List<Sample>>	A list of list of samples in the same order as subids.

1.8.19 deleteExecution

Cancels a running execution.

```
public void deleteExecution(String executionId)
```

Parameter	Type	Description
executionId	String	The execution id of the process to terminate.

1.9 Data types

1.9.1 Sample

A Sample represents a date and a value.

1.9.1.1 Constructor

```
public Sample(org.joda.time.LocalDate date,
              double value)
```

Parameter	Type	Description
date	LocalDate	A date. The LocalDate class is a part of the class library Joda-time.
value	double	A value.

1.9.2 ExecutionStatus

ExecutionStatus enum represents the status of an ongoing simulation run.

1.9.2.1 Values

An ExecutionStatus can have one of four possible values.

Enum constant	Description
DONE	The process is done executing.
NOT_STARTED	The process is in the process of being started but has not yet begun executing.
RUNNING	The process is executing.
UNKNOWN	Status is not known, eg the remote call fails or if there is no such execution.

1.9.3 Scenario

The representation of available scenarios.

1.9.3.1 Constructor

```
public Scenario(String scenarioId,
                String bdate,
                String cdate,
                String edate)
```

Parameter	Type	Description
scenarioId	String	The identifier.
bdate	String	The first date of the simulation. Date format: YYYY-MM-DD
cdate	String	The first date to print. Date format: YYYY-MM-DD
edate	String	The last date of the simulation. Date format: YYYY-MM-DD

1.9.4 TimeSeriesMetadata

A Sample represents a date and a value.

1.9.4.1 Constructor

```
public TimeSeriesMetadata(String timeSeriesId,
                          org.joda.time.LocalDate startDate,
                          org.joda.time.LocalDate endDate,
                          String variable,
                          String unit,
```

```
String resolution)
```

Parameter	Type	Description
timeSeriesId	String	The identifier of the time series.
startDate	org.joda.time.LocalDate	The first date of the time series. The LocalDate class is a part of the class library Joda-time.
endDate	org.joda.time.LocalDate	The last date of the time series. The LocalDate class is a part of the class library Joda-time.
variable	String	The variable it represents.
unit	String	The unit of measure.
Resolution	String	The resolution.

1.10 Web feature service

The WFS is providing feature types with information about subbasins and its upstream subbasins. There are three different feature types:

Feature type 1: basin

subid	id of basin
the_geom	geometri for basin

Feature type 2: upstream_geom

subid	id of basin
upstream_subid	id of upstream basin
basin_geom	geometry of basin
upstream_basin_geom	geometry of upstream basin

Feature type 3: upstream_union_geom

subid	id of basin
basin_geom	geometry of basin
upstream_geom	combined geometry of all upstream basins

The basin feature type has information about each basin and its geometry. The upstream_geom feature type contains pairs of basins and upstream basins. The upstream_union_geom has a geometry for the union of all upstream basins, for each basin.

Here are some examples of querying the feature types:

1. Retrieve basin per id:
http://79.125.2.136:49225/geoserver/wfs?request=GetFeature&typeName=basin&CQL_FILTER=subid=308382&propertyName=subid,the_geom.

1. Retrieve basin per geometry:
http://79.125.2.136:49225/geoserver/wfs?request=GetFeature&typeName=basin&CQL_FILTER=INTERSECT(the_geom,POINT(57.86%2011.87))&propertyName=subid,the_geom

Copyright © SUDPLAN

2. Retrieve upstream basin features per basin id:
http://79.125.2.136:49225/geoserver/wfs?request=GetFeature&typeName=upstream_geom&CQL_FILTER=subid=308382&propertyName=subid,upstream_subid
http://79.125.2.136:49225/geoserver/wfs?request=GetFeature&typeName=upstream_geom&CQL_FILTER=subid=308382&propertyName=subid,upstream_subid,upstream_basin_geom&maxFeatures=5.

3. Retrieve upstream basin features per geometry.
http://79.125.2.136:49225/geoserver/wfs?request=GetFeature&typeName=upstream_geom&CQL_FILTER=INTERSECT(basin_geom,POINT(57.86%2011.87))&propertyName=subid,upstream_subid
http://79.125.2.136:49225/geoserver/wfs?request=GetFeature&typeName=upstream_geom&CQL_FILTER=INTERSECT(basin_geom,POINT(57.86%2011.87))&propertyName=subid,upstream_subid,upstream_basin_geom&maxFeatures=5.

4. Retrieve upstream basins as combined geometry per basin id:
http://79.125.2.136:49225/geoserver/wfs?request=GetFeature&typeName=upstream_union_geom&CQL_FILTER=subid=308382&propertyName=subid,upstream_geom

5. Retrieve upstream basins as combined geometry per geometry:
http://79.125.2.136:49225/geoserver/wfs?request=GetFeature&typeName=upstream_union_geom&CQL_FILTER=INTERSECT(basin_geom,POINT(57.86%2011.87))&propertyName=subid,upstream_geom

Copyright © SUDPLAN