# ADMIRE D6.6 – Report on Pilot Applications Deployment and Platform Evaluation

<table>
<thead>
<tr>
<th>Project Title</th>
<th>ADMIRE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliverable Number</td>
<td>D6.6</td>
</tr>
<tr>
<td>Authorship</td>
<td>Ladislav Hluchy, Ondrej Habala, Maciek Jarka, Martin Šeleng, Viet Tran, Peter Krammer, Branislav Šimo, Michelle Galea, Adrian Mouat, Radek Ostrowski, Ally Hume, Alessandro Spinuso</td>
</tr>
<tr>
<td>Document Filename</td>
<td>ADMIRE-D6.6-Report_On_Pilot_Applications_Deployment_and_Platform_Evaluation.docx</td>
</tr>
<tr>
<td>Document Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Distribution Classification</td>
<td>Public</td>
</tr>
<tr>
<td>Distribution List</td>
<td>ADMIRE Project Team</td>
</tr>
<tr>
<td>Approval List</td>
<td>Michelle Galea, Project Manager, Executive Board</td>
</tr>
</tbody>
</table>
## Document History

<table>
<thead>
<tr>
<th>Personnel</th>
<th>Date</th>
<th>Comment</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ondrej Habala</td>
<td>01/03/2011</td>
<td>Document template</td>
<td>0.1</td>
</tr>
<tr>
<td>Martin Seleng</td>
<td>04/29/2011</td>
<td>FFSC and chapters, Orava DISPEL appendix</td>
<td>0.2</td>
</tr>
<tr>
<td>Branislav Simo</td>
<td>05/02/2011</td>
<td>Orava portal chapter</td>
<td>0.3</td>
</tr>
<tr>
<td>Martin Seleng, Peter Krammer</td>
<td>05/03/2011</td>
<td>Changed FFSC chapter</td>
<td>0.4</td>
</tr>
<tr>
<td>Maciej Jarka</td>
<td>05/19/2011</td>
<td>ACRM application and portal descriptions</td>
<td>0.5</td>
</tr>
<tr>
<td>Branislav Simo</td>
<td>05/30/2011</td>
<td>Executive summary, FFSC progress, integrated Seismo &amp; Astro chapters, DISPELs for RADAR and SVP scenarios</td>
<td>0.6</td>
</tr>
<tr>
<td>Branislav Simo</td>
<td>01/06/2011</td>
<td>Integrated EURExpress-II chapter</td>
<td>0.7</td>
</tr>
<tr>
<td>Branislav Simo</td>
<td>08/06/2011</td>
<td>Added ADMIRE Platform Evaluation chapter</td>
<td>0.8</td>
</tr>
<tr>
<td>Michelle Galea</td>
<td>08/06/2011</td>
<td>Internal review</td>
<td>0.9</td>
</tr>
<tr>
<td>Rob Baxter</td>
<td>08/06/2011</td>
<td>Final edit and signoff</td>
<td>1.0</td>
</tr>
</tbody>
</table>
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Executive Summary</td>
<td>3</td>
</tr>
<tr>
<td>2  Flood Forecasting Simulation Cascade</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Progress since M30 of the Project</td>
<td>4</td>
</tr>
<tr>
<td>2.2 ORAVA</td>
<td>4</td>
</tr>
<tr>
<td>2.2.1 Scenario Description and Goals</td>
<td>4</td>
</tr>
<tr>
<td>2.2.2 Experimental Results</td>
<td>7</td>
</tr>
<tr>
<td>2.2.3 DISPEL Representation of ORAVA</td>
<td>10</td>
</tr>
<tr>
<td>2.3 RADAR</td>
<td>11</td>
</tr>
<tr>
<td>2.3.1 Scenario Description and Goals</td>
<td>11</td>
</tr>
<tr>
<td>2.3.2 Experimental Results</td>
<td>12</td>
</tr>
<tr>
<td>2.3.3 DISPEL Representation of RADAR</td>
<td>15</td>
</tr>
<tr>
<td>2.4 SVP</td>
<td>15</td>
</tr>
<tr>
<td>2.4.1 Scenario Description and Goals</td>
<td>15</td>
</tr>
<tr>
<td>2.4.2 Experimental Results</td>
<td>16</td>
</tr>
<tr>
<td>2.5 Deployment of FFSC within the ADMIRE Testbed</td>
<td>18</td>
</tr>
<tr>
<td>2.6 Conclusion</td>
<td>19</td>
</tr>
<tr>
<td>3  Analytical Platform for Customer Relationship Management (ACRM)</td>
<td>20</td>
</tr>
<tr>
<td>3.1 Progress since PM30</td>
<td>20</td>
</tr>
<tr>
<td>3.2 Churn prediction module</td>
<td>20</td>
</tr>
<tr>
<td>3.3 Cross-selling module</td>
<td>23</td>
</tr>
<tr>
<td>3.3.1 Association rules</td>
<td>23</td>
</tr>
<tr>
<td>3.3.2 Extending data scope</td>
<td>24</td>
</tr>
<tr>
<td>3.3.3 Data preparation</td>
<td>24</td>
</tr>
<tr>
<td>3.3.4 Association rule mining</td>
<td>25</td>
</tr>
<tr>
<td>3.3.5 Association rule representation</td>
<td>27</td>
</tr>
<tr>
<td>4  Automatic Photometric Classification of Quasars</td>
<td>29</td>
</tr>
<tr>
<td>4.1 The data</td>
<td>29</td>
</tr>
<tr>
<td>4.2 Data access and integration workflow</td>
<td>30</td>
</tr>
<tr>
<td>4.3 Performing the data mining</td>
<td>31</td>
</tr>
<tr>
<td>4.4 Conclusion</td>
<td>33</td>
</tr>
<tr>
<td>5  Seismic Noise Interferometry</td>
<td>34</td>
</tr>
<tr>
<td>5.1 Topology of the infrastructure</td>
<td>34</td>
</tr>
<tr>
<td>5.2 Description of the workflow</td>
<td>35</td>
</tr>
<tr>
<td>5.3 User-Defined Processing Elements</td>
<td>38</td>
</tr>
<tr>
<td>5.4 Data Provenance</td>
<td>39</td>
</tr>
<tr>
<td>6  Automatic Gene Annotation of Mice Embryos</td>
<td>42</td>
</tr>
<tr>
<td>6.1 Overview of the annotation process</td>
<td>42</td>
</tr>
<tr>
<td>6.2 Evolution of the use case</td>
<td>43</td>
</tr>
<tr>
<td>6.3 New approach</td>
<td>43</td>
</tr>
<tr>
<td>6.4 New features</td>
<td>43</td>
</tr>
<tr>
<td>7  Application Portals in ADMIRE</td>
<td>46</td>
</tr>
</tbody>
</table>
7.1 Architecture and Technology of ADMIRE Application Portals ......................... 46
7.2 ACRM Churn .............................................................................................................. 47
7.3 ACRM Cross-selling ................................................................................................... 49
7.4 FFSC ORAVA ............................................................................................................. 50
7.5 FFSC RADAR ............................................................................................................. 53
7.6 FFSC SVP .................................................................................................................. 54
8 ADMIRE Platform Evaluation ............................................................................... 56
9 Acronyms .................................................................................................................... 58
10 References .................................................................................................................. 59
Appendix A Description of Datasets And Custom Processing Elements ................. 60
A.1 Data Used in Flood Forecasting Simulation Cascade Scenarios ....................... 60
   A.1.1 Domain Data ........................................................................................................... 60
   A.1.2 MARS ..................................................................................................................... 60
   A.1.3 SVP .......................................................................................................................... 61
   A.1.4 SNOW ...................................................................................................................... 62
   A.1.5 RADAR .................................................................................................................... 63
   A.1.6 SHMU_HYDRO ....................................................................................................... 64
A.2 Custom Processing Elements of Flood Forecasting Simulation Cascade Scenarios ........... 65
Appendix B DISPEL Documents for Application Scenarios ................................. 67
B.1 Cross-selling Scenario ............................................................................................... 67
B.2 ORAVA scenario – Data integration ........................................................................ 69
B.3 ORAVA scenario – Training ..................................................................................... 73
B.4 ORAVA scenario – Prediction .................................................................................. 73
B.5 RADAR Scenario – Motion vector based radar image computation (prediction) with subsequent precipitation forecast ................................................................. 75
B.6 SVP Scenario – Data integration .............................................................................. 76
B.7 SVP Scenario – Training ............................................................................................ 79
B.8 SVP Scenario – Prediction ......................................................................................... 80
B.9 Seismology Scenario – Preprocessing ..................................................................... 81
B.10 Astronomy Scenario – Quasar Classification (Parallel) ........................................ 84
1 Executive Summary

This document is deliverable D6.6 of the ADMIRE project, which reports on the pilot applications status, deployment and platform evaluation at the end of the project.

In the last period of the project we have completed implementation of both pilot applications – the Flood Forecast Simulation Cascade (FFSC or Flood) and Analytical Customer Relationship Management (ACRM) use cases. A set of DISPEL documents exist that implement data processing workflows for individual application scenarios on the ADMIRE Testbed. Those DISPEL documents were created in the ADMIRE Workbench with the help of the ADMIRE Registry which contains the list of available Processing Elements (PEs) deployed in the ADMIRE Testbed. Several new application-specific PEs have been implemented and deployed successfully into the Gateways comprising the distributed ADMIRE infrastructure.

Application workflows can now be submitted for execution both from the Workbench and also from domain-specific application portals. Application portals demonstrate that the use of ADMIRE tools and services can be made transparent for the end user domain experts; the tools can remain hidden when not needed, while their full power and flexibility can be exploited by data mining experts during application development.

In the last period of the project we have successfully exploited the ADMIRE infrastructure with three additional application use cases that were not originally planned: Automatic Photometric Classification of Quasars, Seismic Noise Interferometry and Automatic Gene Annotation of Mice Embryos. The success of these new applications is another demonstration of the power of ADMIRE’s vision and its implementation.
2 Flood Forecasting Simulation Cascade

This chapter describes the final status of the Flood Forecasting Simulation Cascade. We describe the scenarios which were developed, the data available for the scenarios, application-specific processing elements we’ve developed during the course of the project and experimental results.

2.1 Progress since M30 of the Project

In the last period of the project we have updated and finished all three scenarios of the Flood Forecasting Simulation Cascade pilot application.

In particular, we have completed the SVP scenario, involving the creation of new scenario-specific PEs for data filtering, training and prediction, and the creation of DISPEL files implementing the training and prediction workflows of the scenario.

We have fixed a data duplication problem in the ORAVA scenario, which caused NULL values to appear in the integrated data, which in turn resulted in problems during training and prediction phases.

Another new PE has been created for the RADAR scenario which allows users to compute (forecast) future radar image(s) based on motion vectors extracted from sequence of existing images. DISPEL files have been created using this PE in their workflows.

The environmental application scenarios used in ADMIRE are part of the Flood Forecasting and Simulation Cascade application, which has been expanded beyond the borders of flood prediction into a broader environmental domain. There are three scenarios, selected from more than a dozen of candidates provided by hydro-meteorological, water management, and pedological experts in Slovakia. Apart from being useful to the respective domain experts, another main criterion for their selection was their suitability for data mining.

2.2 ORAVA

2.2.1 Scenario Description and Goals

This hydrological scenario has been defined by the Hydrological Service division of the Slovak Hydrometeorological Institute (SHMI). Its goal is to predict the water discharge wave and temperature propagation below the Orava reservoir, one of the largest water reservoirs in Slovakia.

The pilot area covered by the scenario (see Figure 2) lies in the north of Slovakia, and covers a relatively small area. The selected data which influence the scenario’s target variables – the discharge wave propagation, and temperature propagation in the outflow from the Orava reservoir to the river Orava – is depicted in Table 1. The data is gathered from hydro-meteorological sensor networks of several data providers. Figure 1 shows the layout of the sensors below the Orava reservoir. Orange dots represent the sensor network of the Slovak Water Enterprise (SWE), which provides reservoir water temperature and discharge. Red dots show part of the network of hydrological sensors operated by SHMI. These sensors are station in the river Orava and its tributaries, and measure current river temperature and water level. The densest sensor network, depicted by green dots, is the network of precipitation measurement stations, which provide hourly precipitation data. Additionally to these, there are also more complex synoptic sensor stations, depicted in yellow color, which provide precipitation, as well as other climatological parameters. What is not shown in the picture is the mesh of values provided by meteorological radars and by meteorological simulations.
Figure 1: A visualization of actual network of hydro-meteorological sensors in the northern part of Slovakia, around the Orava and Liptovska Mara reservoirs.

Figure 2: The geographical area of the hydrological pilot scenario.
As predictor variables in this scenario, shown in Table 1 we have selected rainfall and air temperature, the discharge volume of the Orava reservoir and the temperature of water in the Orava reservoir. Our target variables are the water height and water temperature measured at a hydrological station below the reservoir. As can be seen in Figure 2, the station directly below the reservoir is no.5830, followed by 5848 and 5880 – these stations are the target sites for which predictions are made. If we run the data mining process in time T, we can expect to have at hand all data from sensors up to this time (first three data lines in Table 1). Future rainfall and temperature can be obtained by running a standard meteorological model. Future discharge of the reservoir is given in the manipulation schedule of the reservoir. The actual data mining targets are the X and Y variables for times after time T (T being current time).

Table 1: Schematic depiction of the predictor variables and targets in the water level and temperature prediction scenario

The ORAVA scenario is implemented as OGS-DAI web services using wrapped Weka\(^1\) functionality for data mining and prediction purposes.

The data retrieval phase is a specialisation of a generic data integration and data mining scenario for spatio-temporal data (see [6] for details). In the case ORAVA, because of the properties of the data used, no temporal or spatial synchronization is necessary (it is partially substituted by the Missing data handling phase activities).

In the data mining phase, the DMI process is divided into two sub-workflows:

- Model training;
- Data predictions.

The model training sub-workflow has the following steps:

1. ProjectionX – which is output from the data integration phase;
2. Data preparation – filtering the data;
   a. Using filters implemented in the Weka library;
   b. Developing our own filters:
      i. Linear trending filter\(^2\);  

\(^1\) [http://www.cs.waikato.ac.nz/ml/weka/](http://www.cs.waikato.ac.nz/ml/weka/)

\(^2\) As measurement of the Orava basin temperature is done in 6 to 12 hours intervals we need to calculate the missing values – as it is a big mass of water the Linear trending filter is relatively good choice to replace the missing measurements.
ii. Replacing of missing values³;
  iii. EpsilonZero filter⁴;
  c. Removing unused (not significant) attributes;
  d. Converting values from one scale to another (Kelvin to degree of Celsius);

3. Classifier building:
   a. Linear Regression;
   b. Bagging;
   c. M5Rules;

4. Classifier evaluation:
   a. Using \( n \)-fold cross-validation;
   b. Using test set (in the above figure we need to split the data in the preparation box into training and test set).

Outputs from the model training sub-workflow are a trained model and a statistical summary (coefficient of correlation, mean absolute error, root mean squared error, relative absolute error, and root relative squared error). The trained model is stored in a repository via a suitable delivery service.

The data prediction sub-workflow has these steps:

- ProjectionX – which is output from the data integration phase (same as above, but using future data from the meteorological models);
- Data preparation – filtering the data (same as in model training sub-workflow);
- Using trained model (de-serialised via FTP service) to make predictions.

Outputs from the data predictions sub-workflows are the predicted values and a statistical summary of the predicted values (same as above in the model training sub-workflow).

2.2.2 Experimental Results

In the ORAVA scenario we have trained several different models with different level of fitting:

- using linear regression (to predict the temperature of the water at the measurement point)
- using neural networks (to predict the temperature of the water at the measurement point)
- using neural networks (to predict the change of the height of the water level at the measurement point)

First some results using linear regression to predict the water temperature. The complete DISPEL files to the data integration, model training and prediction of the height of the water temperature can be found in Appendix B. We use a set of 8760 rows to train the model and also to evaluate it using 10-fold cross-validation. The output function (Equation 1) we have received from the training is:

\[
Water\_temp\_station = 0.6473\times Water\_temp\_Orava + 0.0239\times Air\_temp\_Orava - 0.0359\times Rainfall\_Orava - 0.0055\times Outflow\_Orava - 0.0418\times Rainfall\_station + 0.0117\times Air\_temp\_station - 0.0503\times Flow\_station + 2.4324
\]

Equation 1 Trained model - linear equation

We have also tried to train a model using neural networks - multilayer perceptron without hidden layers. The activation function was sigmoid function (input layer) and linear function (output layer).

³ As the Weka library doesn’t have a filter which will deal with missing values, we have developed our own – as there are missing values in rainfall data, we need to replace it with 0 value (no rain – no rainfall).
⁴ EpsilonZero filter replaces values which are less than given epsilon with zeros.
Our neural network consists from five perceptrons. In fact neural networks are very popular in meteorological modeling and also from the comparison between models one can tell (Table 2) that a neural network is better choice as linear regression. Table 3 shows an example of a few predicted and actual (validation) values of the trained models. We can see that the correlation coefficient is really high and mean absolute and root mean squared errors are relatively low, so we conclude that our models are well trained for the data we have, and we can use it for predictions in the future. From that table we can also see that neural network model gives us little bit accurate results, but the training time is much longer.

<table>
<thead>
<tr>
<th>Model\Properties</th>
<th>Linear regression</th>
<th>Multilayer perceptron</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation coefficient</td>
<td>0.9639</td>
<td>0.9821</td>
</tr>
<tr>
<td>Mean absolute error</td>
<td>1.1791</td>
<td>0.7748</td>
</tr>
<tr>
<td>Root mean squared error</td>
<td>1.4607</td>
<td>1.0386</td>
</tr>
<tr>
<td>Relative absolute error</td>
<td>23.8739 %</td>
<td>15.6884 %</td>
</tr>
<tr>
<td>Root relative squared error</td>
<td>26.609 %</td>
<td>18.9195 %</td>
</tr>
<tr>
<td>Total Number of Instances</td>
<td>8760</td>
<td>8760</td>
</tr>
</tbody>
</table>

Table 2 Selected parameters of the trained data mining models (prediction of the water temperature) of the ORAVA scenario

In the last model training we have used 8735 records to predict the change of the water level, based on the current situation. The advantage of this approach is that we can predict the change of the water level for several hours to the future (it just depends on the meteo model accuracy). We have used the next 6 input parameters:

\[
\text{Rainfall}_{\text{station}}[S+1], \text{Flow}_{\text{station}}[X], \text{Outflow}_{\text{Orava}}[D], \text{Outflow}_{\text{Orava}}[D+1]-\text{Outflow}_{\text{Orava}}[D], \ln(\text{Outflow}_{\text{Orava}}[D]), \sqrt{\text{Outflow}_{\text{Orava}}[D]}
\]

And output is the change of the water level:

\[
\text{Flow}_{\text{station}}[X+1]-\text{Flow}_{\text{station}}[X]
\]

The activation function was again a sigmoid function and the statistical description of the trained model can be found in the Table 4. Some results with predicted and real values can be found in the Table 5.
Validation data (temperature)

<table>
<thead>
<tr>
<th>Validation data</th>
<th>Linear regression model</th>
<th>Multilayer perceptron model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Predicted data</td>
<td>Error</td>
</tr>
<tr>
<td>11.6</td>
<td>13.071</td>
<td>1.471</td>
</tr>
<tr>
<td>15.2</td>
<td>14.335</td>
<td>-0.865</td>
</tr>
<tr>
<td>6.4</td>
<td>7.614</td>
<td>1.214</td>
</tr>
<tr>
<td>0.7</td>
<td>2.284</td>
<td>1.584</td>
</tr>
<tr>
<td>11.7</td>
<td>10.948</td>
<td>-0.752</td>
</tr>
<tr>
<td>14.3</td>
<td>16.526</td>
<td>2.226</td>
</tr>
<tr>
<td>15.6</td>
<td>12.891</td>
<td>-2.709</td>
</tr>
<tr>
<td>15.7</td>
<td>12.838</td>
<td>-2.862</td>
</tr>
<tr>
<td>0.8</td>
<td>1.752</td>
<td>0.952</td>
</tr>
<tr>
<td>15.8</td>
<td>15.188</td>
<td>-0.612</td>
</tr>
<tr>
<td>15.4</td>
<td>16.553</td>
<td>1.153</td>
</tr>
<tr>
<td>14.9</td>
<td>12.795</td>
<td>-2.105</td>
</tr>
<tr>
<td>15.4</td>
<td>15.660</td>
<td>0.260</td>
</tr>
<tr>
<td>0.7</td>
<td>2.318</td>
<td>1.618</td>
</tr>
<tr>
<td>15.0</td>
<td>12.628</td>
<td>-2.372</td>
</tr>
<tr>
<td>14.8</td>
<td>16.692</td>
<td>1.892</td>
</tr>
<tr>
<td>10.1</td>
<td>8.530</td>
<td>-1.570</td>
</tr>
<tr>
<td>7.6</td>
<td>8.311</td>
<td>0.711</td>
</tr>
<tr>
<td>12.9</td>
<td>13.699</td>
<td>0.799</td>
</tr>
<tr>
<td>10.3</td>
<td>7.390</td>
<td>-2.910</td>
</tr>
</tbody>
</table>

Table 3: Sample of the actual (validation) and predicted data for both trained models (prediction of the water temperature)

Model\Properties

<table>
<thead>
<tr>
<th>Model\Properties</th>
<th>Multilayer perceptron</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation coefficient</td>
<td>0.9816</td>
</tr>
<tr>
<td>Mean absolute error</td>
<td>0.4105</td>
</tr>
<tr>
<td>Root mean squared error</td>
<td>0.9673</td>
</tr>
<tr>
<td>Relative absolute error</td>
<td>30.5869 %</td>
</tr>
<tr>
<td>Root relative squared error</td>
<td>19.2384 %</td>
</tr>
<tr>
<td>Total Number of Instances</td>
<td>8735</td>
</tr>
</tbody>
</table>

Table 4: Selected parameters of the trained data mining models (prediction of the water level change) of the ORAVA scenario.
Table 5: Sample of the actual (validation) and predicted data for trained model (prediction of the water level change)

<table>
<thead>
<tr>
<th>Validation data (change of the water level)</th>
<th>Multilayer perceptron model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Predicted data</td>
</tr>
<tr>
<td>1</td>
<td>0.887</td>
</tr>
<tr>
<td>-1</td>
<td>-0.577</td>
</tr>
<tr>
<td>-1</td>
<td>-1.568</td>
</tr>
<tr>
<td>0</td>
<td>0.036</td>
</tr>
<tr>
<td>0</td>
<td>-0.296</td>
</tr>
<tr>
<td>-31</td>
<td>-35.6</td>
</tr>
<tr>
<td>1</td>
<td>0.68</td>
</tr>
<tr>
<td>2</td>
<td>1.873</td>
</tr>
<tr>
<td>1</td>
<td>1.013</td>
</tr>
<tr>
<td>0</td>
<td>0.071</td>
</tr>
<tr>
<td>-4</td>
<td>-3.909</td>
</tr>
<tr>
<td>0</td>
<td>-0.232</td>
</tr>
<tr>
<td>17</td>
<td>14.746</td>
</tr>
<tr>
<td>-1</td>
<td>-1.237</td>
</tr>
<tr>
<td>0</td>
<td>-0.235</td>
</tr>
<tr>
<td>-10</td>
<td>-9.547</td>
</tr>
<tr>
<td>0</td>
<td>-0.07</td>
</tr>
</tbody>
</table>

2.2.3 DISPEL Representation of ORAVA

The scenario can be divided into three phases: data integration, training and prediction, as seen in Error! Reference source not found.. There is one single DISPEL file for whole scenario; however, the file is quite long, complicated and it is difficult to understand and debug. Therefore, for convenience, we created three separate DISPEL files for each phase. The first DISPEL file integrates required data from data sources and saves results to a repository. The second one reads the integrated data from the repository, train model on the data and save trained model again in the repository. The last one uses the saved model from previous training, reads another integrated data set from the repository and makes predictions.

The data integration DISPEL reads rainfall and air temperature from the meteorological data, water temperature and flow rate from the water reservoir data, and water level and temperature from the water stations data.

Meteorology data are stored in GRIB file format, a common data format for meteorology. Reading data from GRIB files is realized in two steps. In the first step, we have to select which GRIB files we need to read data from according to date/time and type of data (gribFileChooser, line 25, 28, 33 in the DISPEL file), and from which cell in the file we need to read the data according to location (gribCoordsSelector, line 26, 29, 37). From this information, we can use the GribCellValue to read the required data from the chosen cell from the selected GRIB file (lines 24, 39, 40). Data from water reservoir and water stations are stored in a database so we can read data from them using the standard SQLQuery PE from OGSA-DAI (lines 139-158).

The data must be cleaned and transformed before integration. There are several filters in the DISPEL file: cleaning numerical errors from meteorological simulation (ZeroEpsilon filter, lines 101-103), interpolating missing data (LinearTrend filter, lines 122-124), temperature conversion from Kelvin to degrees Celsius (TupleArithmeticProject, lines 71-74). Finally, the data are merged (OrderedTuplesMerge, lines 162-190) and stored to the repository (DeliverToFTP, lines 206-209).
The DISPEL file for training first reads data from the repository (ObtainFromFTP, lines 14-18), then trains the model (Classify, lines 19-23) and stores the trained model to repository (lines 29-34). The DISPEL file for prediction first reads data and model from repository (ObtainFromFTP, lines 21-30), then make prediction (BuildClassifierLinearRegression, lines 19-23) and deliver prediction data.

2.3 RADAR

2.3.1 Scenario Description and Goals

The usual methods of meteorological prediction, whose results we see daily in weather reports, are optimised for long-term prediction up to 48 hours, and are not suitable for the prediction of flash floods and nowcasting. These problems are tackled in our second pilot scenario. This experimental scenario tries to predict the movement of moisture in the air from a series of radar images (see Figure 3). Weather radar measures the reflective properties of air, which are transformed to potential precipitation before being used for data mining. An example of an already-processed radar sample (with the reflection already re-computed to millimeters of rainfall accumulated in an hour) can be seen in Figure 3.

The scenario once again uses both historical precipitation data (measured by sensors maintained by SHMI) and weather predictions computed by a meteorological model. Additionally to these, SHMI has provided several years’ worth of weather radar data (already transformed to potential precipitation).

The extrapolation of observations in time has been demonstrated to be more accurate up to several hours in numerical weather prediction, than the prediction of physics-based models. The purpose of this work is to investigate various approaches to extrapolating precipitation in time. The purpose of the scenario is to allow very short-term prediction of precipitation using radar imagery and historical values of measured precipitation. Input of the scenario is a time-series of radar imagery (reflectivity and potential 15min accumulated precipitation) and historical values of 24h accumulated precipitation measured by ground sensor network. The output of the scenario is a set of predicted values of precipitation for the whole territory of Slovakia. The predicted precipitation can be directly verified against the potential precipitation or against the ground based sensors.

A secondary output is the motion vector time-series produced from the radar images. The process of the scenario is as follows:

1. Read two successive radar images
2. Find motion vectors as a combination of these two images. The simple method to compute motion vectors is to use moving frames which correlation computed inside frames in between images.
3. Repeat steps 1 and 2 to transform the radar image time-series to motion vector time-series
4. Smooth motion vectors over some time period (30min - to 2 hours)
5. Assume linear translation of precipitation at time t with smoothed motion vector computed over interval with end at time instant t. We can compute 15min accumulated prediction up to 2 hours from time t (every time t we produce prediction t+15min, t+30min, t+45min, t+120min)
6. Compute error of predicted 15min accumulated precipitation using the potential 15min accumulated precipitation. In this way we can compute time series of error for different prediction ranges.

An alternate approach is to use a data mining technique with the model based on analogy. We look for similar sequence of images and we assume that the prediction is simply analogous to one that has occurred already in the past. This approach can be significantly improved probably by proper calibration of radar images against the ground based observations as follows:

1. Train model using the sequence of potential 15min accumulated precipitation (or by some diagnostics based on precipitation)
2. Produce prediction of 15min accumulated precipitation using trained model
3. Verify predicted data against the potential 15min accumulated precipitation

Figure 3: An example of weather radar image with potential precipitation

2.3.2 Experimental Results

The RADAR scenario is designed for the short-term prediction of rainfall based on the reflection of radar signals from water vapour in the atmosphere. A matrix of reflection is created from radar frames, which are supplied by the Slovak Hydro-Meteorological Institute, and actual data are available every 15 minutes.

The rainfall information is available from synoptic stations with an hourly period - cumulative data for the preceding hour. The reflection matrix represents the quantity of water in the atmosphere; hence rainfall is significantly correlated with radar reflection.

Figure 4: Network of synoptic stations in Slovakia
Scaling radar data from 15 minutes to 60 minutes period is important, because we need all data in the same time scale.

Our purpose is to create a model which will describe rainfall, suitable for all synoptic stations. Therefore, the training process is realized on data from all measured stations at the same time. The output of the model is the rainfall for a specified synoptic station; input is the reflectivity matrix in the area around the synoptic station.

The advantage of isotonic regression is its suitability for significantly nonlinear models, such as meteorological ones. Another reason for its use is the fact that there are other variables present which significantly influence the meteorological process, but are unknown to us (we don’t have the data which describe them).
Figure 6: Graphical representation of the short-term weather prediction scenario data flow. Example of prediction at time instant t. Motion vectors are averaged using last three motion vectors computed and then prediction is computed using precipitation at time instant t and averaged motion vectors. Precipitation is computed up to thirty minutes with a fifteen-minute step. Error of prediction can be directly computed using potential precipitation time series.

<table>
<thead>
<tr>
<th>index of cut points</th>
<th>prediction (rainfall)</th>
<th>cut point (reflective)</th>
<th>index of cut points</th>
<th>prediction (rainfall)</th>
<th>cut point (reflective)</th>
<th>index of cut points</th>
<th>prediction (rainfall)</th>
<th>cut point (reflective)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.01</td>
<td>1.78</td>
<td>15</td>
<td>0.23</td>
<td>96.91</td>
<td>29</td>
<td>1.35</td>
<td>355.91</td>
</tr>
<tr>
<td>2</td>
<td>0.03</td>
<td>1.84</td>
<td>16</td>
<td>0.28</td>
<td>97.47</td>
<td>30</td>
<td>1.40</td>
<td>377.19</td>
</tr>
<tr>
<td>3</td>
<td>0.03</td>
<td>8.28</td>
<td>17</td>
<td>0.30</td>
<td>129.63</td>
<td>31</td>
<td>1.52</td>
<td>381.78</td>
</tr>
<tr>
<td>4</td>
<td>0.03</td>
<td>16.97</td>
<td>18</td>
<td>0.33</td>
<td>129.72</td>
<td>32</td>
<td>2.13</td>
<td>395.31</td>
</tr>
<tr>
<td>5</td>
<td>0.03</td>
<td>24.28</td>
<td>19</td>
<td>0.42</td>
<td>147.94</td>
<td>33</td>
<td>2.23</td>
<td>399.16</td>
</tr>
<tr>
<td>6</td>
<td>0.03</td>
<td>36.91</td>
<td>20</td>
<td>0.44</td>
<td>168.59</td>
<td>34</td>
<td>2.28</td>
<td>447.06</td>
</tr>
<tr>
<td>7</td>
<td>0.05</td>
<td>37.53</td>
<td>21</td>
<td>0.50</td>
<td>187.13</td>
<td>35</td>
<td>2.60</td>
<td>447.69</td>
</tr>
<tr>
<td>8</td>
<td>0.05</td>
<td>38.72</td>
<td>22</td>
<td>0.51</td>
<td>187.47</td>
<td>36</td>
<td>2.60</td>
<td>467.66</td>
</tr>
<tr>
<td>9</td>
<td>0.06</td>
<td>44.53</td>
<td>23</td>
<td>0.62</td>
<td>211.56</td>
<td>37</td>
<td>2.98</td>
<td>515.19</td>
</tr>
<tr>
<td>10</td>
<td>0.07</td>
<td>59.03</td>
<td>24</td>
<td>0.72</td>
<td>268.38</td>
<td>38</td>
<td>3.75</td>
<td>625.56</td>
</tr>
<tr>
<td>11</td>
<td>0.08</td>
<td>61.16</td>
<td>25</td>
<td>0.93</td>
<td>281.28</td>
<td>39</td>
<td>4.93</td>
<td>665.41</td>
</tr>
<tr>
<td>12</td>
<td>0.10</td>
<td>61.78</td>
<td>26</td>
<td>1.00</td>
<td>297.72</td>
<td>40</td>
<td>5.24</td>
<td>901.25</td>
</tr>
<tr>
<td>13</td>
<td>0.14</td>
<td>81.59</td>
<td>27</td>
<td>1.14</td>
<td>314.47</td>
<td>41</td>
<td>5.40</td>
<td>934.41</td>
</tr>
<tr>
<td>14</td>
<td>0.19</td>
<td>89.22</td>
<td>28</td>
<td>1.26</td>
<td>344.59</td>
<td>42</td>
<td>6.30</td>
<td>971.5</td>
</tr>
</tbody>
</table>

Table 6: Results of the RADAR experimental data mining

Model: Isotonic Regression, table of cut points
Model performance:

Validation: 10 – Cross fold

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation coefficient</td>
<td>0.4593</td>
</tr>
<tr>
<td>Mean absolute error</td>
<td>0.1105</td>
</tr>
<tr>
<td>Root mean squared error</td>
<td>0.5490</td>
</tr>
<tr>
<td>Total number of instances</td>
<td>89,746</td>
</tr>
</tbody>
</table>

Table 7: Selected parameters of the trained data mining models of the RADAR scenario

Hydrometeorological performance:
Probability of detection with threshold 0.3 and 0.6 mm rainfall per hour.

\[
POD_{0.3} = 63.87\% \\
POD_{0.6} = 56.22\%
\]

Miss rate with threshold 0.3 and 0.6 mm rainfall per hour.

\[
MR_{0.3} = 1.85\% \\
MR_{0.6} = 1.58\% 
\]

2.3.3 DISPEL Representation of RADAR

Radar data are stored in a binary format akin to bitmaps, one byte for each pixel. The radar data are read in two steps: first we need to select which files to read according to date/time (GetFileNames PE, lines 12-15) then read the actual data from chosen location (ReadData PE, lines 16-19). Data from meteorological stations are stored in relational database (line 20-24). The data are merged according to time (OrderedTupleMerge, lines 25-31) then converted to text format for delivering (lines 32-36).

2.4 SVP

2.4.1 Scenario Description and Goals

This scenario is the most complex of all scenarios deployed in the context of ADMIRE. It uses the statistical approach to do what the FFSC application did before ADMIRE – predict floods.

The reasons we decided to perform this experiment are mainly the complexity of simulation of floods by physical models when taking into account more of the relevant variables, and the graceful degradation of results of the data mining approach when facing incomplete data – in contrast to the physical modeling approach, which usually cannot be even tried without having all the necessary data.

For predicting floods, we have been equipped with 10 years of historical data from the Vah cascade of waterworks by the Slovak Water Enterprise, 9 years of meteorological data (precipitation, temperature, wind) computed by the ALADIN model at SHMI, hydrological data from the river Vah, again by SHMI, and additionally with measured soil capacity for water retention, courtesy of our partner Institute of Hydrology of the Slovak Academy of Sciences. We base our efforts on the theory, that the amount of precipitation, which actually reaches the river basin and contributes to the water level of the river, is influenced by actual precipitation and its short-term history, water retention capacity of the soil, and to lesser extent by the evapotranspiration effect.

In current experiment, we try to predict water flow into Orava water reservoir at the beginning of the Orava river, the pilot site of the first scenario as shown in Figure 1. This inflow is very important for waterworks manipulation and also one of the basic factors for flood prediction in the downstream. Once the model for inflow to reservoir is created, we can use its outputs for further models at downstream like in the first scenario.
2.4.2 Experimental Results

Data sets with very similar structure are needed for both prediction and training of models, so the integration process is common to both steps. The historical air temperatures and rainfall at Orava reservoir are obtained from waterworks data provided by Slovak Water Enterprise. Future temperatures and rainfalls can be calculated by the ALADIN model. As data used in this scenario have different periods (from hourly for air temperature to weekly for snow level), data aggregation and interpolation must be used for time synchronization.

Filters like “delete invalid rows filter” or “correction filter” are applied in the next phase, which provides the necessary data integrity and guarantees unique date. Before building a classifier it is necessary to preprocess the integrated data, which primarily consists of choosing the required columns. The predictor and target variables are shown in Table 11. The target variable, water flow into the reservoir, is predicted using two models. The first, a snow-melting model, is used to predict the snow level using the temperature, rainfall and the snow level of the previous day. This model is based on linear regression. The second, a run-off model is used to predict the flow into the reservoir from rainfall, snow-melting levels from the previous model, and inflow from the previous day. The second model uses a decision tree, with the M5P training algorithm. The characteristics of these models are show in Table 8, Table 9 and Table 10. As a validation method we used 10-fold cross validation. These two models were trained on historical 10-years data, and were stored into repository. Prediction process is realized by sequent apply classifiers - snow model and flow model, for each predicted day. Days in prediction are processing from oldest to last, and first day must contain valid columns into item snow and inflow (Table 11).

<table>
<thead>
<tr>
<th></th>
<th>Perceptron Neural Network</th>
<th>Gaussian Process</th>
<th>Linear Regression</th>
<th>Decision Tree M5P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation coefficient</td>
<td>0.8810</td>
<td>0.8469</td>
<td>0.8079</td>
<td>0.8899</td>
</tr>
<tr>
<td>Mean absolute error</td>
<td>7.0577</td>
<td>6.9821</td>
<td>8.3816</td>
<td>5.2562</td>
</tr>
<tr>
<td>Relative absolute error</td>
<td>40.5821%</td>
<td>40.1472%</td>
<td>48.1942%</td>
<td>30.2231%</td>
</tr>
<tr>
<td>Root relative squared error</td>
<td>48.6547%</td>
<td>53.4747%</td>
<td>58.8616%</td>
<td>45.5415%</td>
</tr>
</tbody>
</table>

Table 8: Characteristics of inflow prediction model for 10-fold cross validation. Inflow value is in m$^3$/s

<table>
<thead>
<tr>
<th></th>
<th>N = 10</th>
<th>N = 20</th>
<th>N = 25</th>
<th>N = 50</th>
<th>N = 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation coefficient</td>
<td>0.8899</td>
<td>0.8933</td>
<td>0.8855</td>
<td>0.8937</td>
<td>0.8934</td>
</tr>
<tr>
<td>Mean absolute error</td>
<td>5.2562</td>
<td>5.1253</td>
<td>5.2484</td>
<td>5.0973</td>
<td>5.0908</td>
</tr>
<tr>
<td>Relative absolute error</td>
<td>30.2231%</td>
<td>29.4869%</td>
<td>30.2017%</td>
<td>29.3317%</td>
<td>29.2915%</td>
</tr>
<tr>
<td>Root relative squared error</td>
<td>45.5415%</td>
<td>44.9218%</td>
<td>46.4373%</td>
<td>44.8306%</td>
<td>44.9086%</td>
</tr>
</tbody>
</table>

Table 9: N-fold cross validation; Decision tree model (M5P) for inflow prediction

---

ADMIRE-REPORT ON PILOT APPLICATIONS DEPLOYMENT AND PLATFORM EVALUATION D6.6

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation coefficient</td>
<td>0.9981</td>
</tr>
<tr>
<td>Mean absolute error</td>
<td>3.3237</td>
</tr>
<tr>
<td>Root mean squared error</td>
<td>4.9055</td>
</tr>
<tr>
<td>Relative absolute error</td>
<td>5.0256%</td>
</tr>
<tr>
<td>Root relative squared error</td>
<td>6.2128%</td>
</tr>
<tr>
<td>Total Number of Instances</td>
<td>1127</td>
</tr>
</tbody>
</table>

Table 10: Characteristics of snow prediction model (Linear regression); 10-fold cross validation. Snow value represents millions m$^3$ of water contained in snow.

<table>
<thead>
<tr>
<th>Date</th>
<th>Temperature</th>
<th>Rainfall</th>
<th>Snow_prev</th>
<th>Snow</th>
<th>Inflow_prev</th>
<th>Inflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>t-1</td>
<td>E(t-1)</td>
<td>R(t-1)</td>
<td>S(t-1)</td>
<td></td>
<td></td>
<td>F(t-1)</td>
</tr>
<tr>
<td>t</td>
<td>E(t)</td>
<td>R(t)</td>
<td>P(t)</td>
<td>S(t)</td>
<td>I(t)</td>
<td>F(t)</td>
</tr>
<tr>
<td>t+1</td>
<td>E(t+1)</td>
<td>R(t+1)</td>
<td>P(t+1)</td>
<td>S(t+1)</td>
<td>I(t+1)</td>
<td>F(t+1)</td>
</tr>
<tr>
<td>t+2</td>
<td>E(t+2)</td>
<td>R(t+2)</td>
<td>P(t+2)</td>
<td>S(t+2)</td>
<td>I(t+2)</td>
<td>F(t+2)</td>
</tr>
<tr>
<td>t+3</td>
<td>E(t+3)</td>
<td>R(t+3)</td>
<td>P(t+3)</td>
<td>S(t+3)</td>
<td>I(t+3)</td>
<td>F(t+3)</td>
</tr>
<tr>
<td>t+4</td>
<td>E(t+4)</td>
<td>R(t+4)</td>
<td>P(t+4)</td>
<td>S(t+4)</td>
<td>I(t+4)</td>
<td>F(t+4)</td>
</tr>
</tbody>
</table>

Table 11: Input and target variables in flow prediction process

The predictor and target variables are shown in Table 11. The target variable, water flow to the reservoir, is predicted using the two models. These two models were trained on historical 10-years data and stored in a repository. The prediction process is realized by sequentially-applied classifiers – snow model and flow model – for each predicted day. During prediction, days are processed from earliest to latest, and the first day must contain valid columns for snow and inflow (Table 11). Snow and inflow values into columns “Snow_prev” and “Inflow_prev” are taken from previous day data:

\[
P(t) = S(t-1)
\]

\[
I(t) = F(t-1)
\]

New values for snow and inflow will be counted using the two created models. The function $f$ represents the linear regression snow model. The function $h$ represents the M5P decision tree model for inflow prediction.

\[
S(t) = f(P(t), R(t), E(t))
\]

\[
F(t) = h(I(t), S(t), E(t), R(t))
\]

A similar process is repeated for each predicted day. By this technique, we can make predictions for 3-5 days ahead. The algorithms begin to deviate too much beyond about 5 days. In Table 12 we present and compare the measured and predicted values for 1 day ahead.
2.5 Deployment of FFSC within the ADMIRE Testbed

At the moment, the ORAVA, RADAR and SVP scenarios are deployed at II SAS. The Gateway and OGSA-DAI server are installed on the server hudson.ui.sav.sk. The PEs mentioned are deployed on the OGSA-DAI server. Additionally, the descriptions of the PEs have been added to the Registry of the Gateway (using the DISPEL document ProcessingElementRegistration.dispel).

Databases are deployed on the virtual server hicks.ui.sav.sk. These databases must be properly configured as data resources for access via SQL. These resources are also added to Gateway configuration file for mapping the correct OGSA-DAI server to the data resources.

Binary data (meteorological and radar data) are stored as files on hudson.ui.sav.sk. The locations of these data are configured as parameters for the various file selection PEs.

The FFSC part of the Testbed thus consists of four virtual servers, hosted at II SAS:

- hicks.ui.sav.sk – stable version of ADMIRE middleware, 1st copy of FFSC database
- hudson.ui.sav.sk – experimental version of ADMIRE middleware, 2nd copy of FFSC database; main repository of FFSC repository, accessible by PEs from DISPEL documents
- vasquez.ui.sav.sk – backup server for ADMIRE middleware and distributed data integration experiments
- drake.ui.sav.sk – general backup server for FFSC
2.6 Conclusion

The DISPEL language has allowed us to describe the processes in each of the three environmental risk scenarios at a high level of abstraction, independently of any low-level concerns regarding the underlying enactment engines, databases or any consideration of the distributed environment. Our experiments have made use of several interconnected gateways, which together provide all the necessary data, processing elements and visualisation tools which our scenarios require.

This novel approach allows us easily to extend the FFSC infrastructure to new data providers, by deploying a Gateway at the site of the new provider, and registering it with the other Gateways. Then, when a data analysis expert creates a DISPEL document that makes use of one of the capabilities provided by this gateway, it can be accessed and integrated automatically into the overall knowledge discovery workflow.

Additionally this approach allows us to use remote high-performance data mining tools, or to access other data storage facilities.

This model provides a clear separation of responsibilities between data-intensive engineers, data analysis experts, and the domain experts of the application. The underlying infrastructure and gateway network is managed by the data-intensive engineers. The data analysis experts speak DISPEL and create full knowledge discovery workflows which use the infrastructure without needing to understand it; in turn these workflows are used by the domain experts via specialized domain-specific portals.

Apart from separating the concerns of the involved stakeholders, this approach also separates the technology into fairly independent layers, and so a once-tuned DISPEL document will work even when the infrastructure changes significantly, provided that infrastructure is still capable of providing all of the processing elements referenced by the document.

This approach also provides for a reasonable amount of fault tolerance. If one data centre becomes unavailable, it may conceivably be replaced transparently by a different one, without the final users of our product ever knowing it happened. Some centers and gateways are, of course, irreplaceable in a given network (primary data storage centers for instance), but data filters may be deployed at several locations to enhance redundancy. There may be also several HPC facilities available to a given user, so the temporary inaccessibility of one of them is no issue – the DISPEL description of the required data-oriented solution is entirely agnostic of such things.
3 Analytical Platform for Customer Relationship Management (ACRM)

This chapter describes the final version of two Analytical CRM modules developed within the ADMIRE project – churn prediction and cross-selling. It contains technical details, deployment plans and experience reached during the project. At the end of the chapter, we describe how effort during development can be reused in various types of analyses and data mining applications.

3.1 Progress since PM30

The additional three months gave us the opportunity to extend both modules with some innovations that were not planned, and which resulted in novel solutions. Moreover, changes in the ADMIRE architecture, language and tools also influenced the development and deployment of the modules and allowed us to improve them to create complex data mining and integration applications.

3.2 Churn prediction module

Compared to the previous reporting period, there were no changes to the core of the churn prediction module. The main task here has been to build the best possible predictive model, that can be used to find out which current customers are likely to leave company services. From a data mining point of view, it is a classical data mining activity of classification, which can be divided into training and evaluation processes. Details of the workflow we designed for both activities are presented in previous deliverable documents from WP6 ([1][2][3]).

In D6.5 [8], we presented the idea of large scale experiments with which we planned to evaluate the ADMIRE architecture. These tests were successful and we performed long-lasting training processing on large volumes of synthetic data. The streaming model was able to handle a distributed and large network of processing nodes.

The final period of development of the module has been spent mostly on a user interface of the application that we call a Portal. The Portal details are described in Section 7. Here we focus on changes to the churn prediction workflow that had to be made to cooperate with these applications.

The basic idea of the workflow designed and configured by a data mining expert in the workbench is to train and evaluate a predictive model. The general chain of activities is presented in Figure 7.

In the data access phase, the SQL queries addressed to the CRM operational database are performed. To improve performance of the data access phase, we used several independent queries instead of one large and complicated query. Then, these streams of data are joined by the appropriate PE. For the data preparation phase, a data mining expert can specify any restriction for the incoming data stream, which includes horizontal and vertical modifications of the data. In the data split phase, the stream is horizontally divided into n+1 substreams, where n indicates the number of models to train – typically one. The last substream is the test substream to evaluate the quality of classification. The model training phase consists of classification building activities. Depending on the number of streams, an appropriate number of BuildClassifier/BuildIterativeClassifier PE’s are created. Using this scenario we tested a large number of algorithms provided by the WEKA and MOA libraries. The most interesting experiments have been performed with algorithms from the latter one, because they were able to handle long data streams and produce results while processing.
The result of the workflow is a single or multiple training models\(^6\) delivered to the Repository together with model metadata. The metadata contains a high level description that can be understood by domain experts while using the application and accessing these results. In churn prediction, domain experts use serialized binary models to predict churning capability for any customer. The prediction phase may also be performed within DMI workflow executed from the portal instead of the workbench. Here, the process is easier as it does not process large portions of data, but binary models and sample data instead (Figure 8).

Here, in the data access phase, customers’ data is being loaded from file or typed input (web forms). Typically, in the application, the data should contain at least data attributes that the classifier has been trained on. Otherwise, the missing values will be replaced with default values and this can significantly influence the classification result. In the model access phase, selected predictive models are retrieved from the Repository and then transformed from binary representation to the form usable in the workflow. The classification phase simply uses a model (or multiple models) to classify incoming data. Here, the classification results are values of the prediction for every data example accessed. In the simplest case, it is 0 for not churned and 1 for churned. In this process, no pattern extraction from the data is performed, so that it is simple and quick. The domain expert can perform them automatically from the Portal.

---

\(^6\) Multiple models can be trained in Random Forest extension of the workflow
Unlike a data mining expert, in the churn prediction scenario, the domain expert does not need to possess any knowledge or experience of the DISPEL language to perform the prediction. The classification workflow is “hidden” under the logic of the application. This is possible because of an exploitation of advanced DISPEL features, which allows a workflow to be parameterized, and new values to be submitted. The workflow logic is wrapped in a single DISPEL function with the following header:
The role of application developer is therefore to provide a capability of model browsing and data input for the domain expert. Values received by the application are then replaced in the DISPEL function call. No PE’s have to be added or removed, so only basic text replacement in the file is necessary. The described architecture of the churn prediction module is depicted in Figure 9 above.

3.3 Cross-selling module

The last few months we spent mostly on development of the cross-selling module, where the business goal is to find out hints about additional products or services to be provided to potential customers. In other words, in this scenario, the analysts want to retrieve frequent patterns from historical data, of which products are often sold together. The basis of this scenario is association rule mining, which describes which products or groups are often sold together, for example:

Roaming = TRUE & GSM_Prepaid = TRUE => Voice_mail = TRUE

3.3.1 Association rules

The general ideas behind association rules can be captured as follows:

- Item – singular product in the database of transactions
- Transaction – set of items that have been bought together by one customer. In our scenario we treat a customer’s purchase history as one transaction
- Support – minimal frequency of the itemset to appear within database of transactions. If item A and item B appear together in exactly 10% of all transactions, we say, that its relative support is 0.1.
- Itemset – a set of items
- Frequent itemset – an itemset with support above a set minimum threshold
- Body of the rule – itemset on the left side of the rule
- Head of the rule – itemset on the right side of the rule

3.3.1.1 Association rule quality

In this scenario, we work only over a training data set, so unlike in churn prediction, the results are not evaluated against a test dataset. On the other hand, we mark retrieved rules with a few specific measures:

**Support** – relative frequency of sum of itemsets from left and right side of the rule:

\[
\text{Sup}(A \rightarrow B)
\]

**Confidence** – support of the rule divided by support of the body of the rule:

\[
\text{Conf}(A \rightarrow B) = \frac{\text{Sup}(A \rightarrow B)}{\text{Sup}(A)}
\]

\[
\text{Lift}(A \rightarrow B) = \frac{\text{Sup}(A \rightarrow B)}{\text{Sup}(B)}
\]
**Leverage** – subtraction of total support of the rule and support of the head of the rule multiplied by support of the body:

\[ \text{Lev}(A \rightarrow B) = \text{Sup}(A \rightarrow B) - \text{Sup}(A) \times \text{Sup}(B) \]

All of the listed measures are very useful in the analysis. For instance, a large value of the support does not necessarily indicate that the rule is interesting, as it can describe a very obvious pattern. Sometimes, rules with a very low support value, but with a large lift value are more interesting – they can describe purchases that do not happen frequently, but when they do the items included are very often sold together in these kinds of transactions.

Depending on the analysis view, retrieved rules can be sorted by any of these measures.

### 3.3.2 Extending data scope

At the beginning, we mentioned that we use a transactions database as a historical data source for rules retrieval. In fact, in our scenario, we integrate two logically different types of data – transactions and customer details. In other words, rules can also contain some information about customers in the body of the rule, for example, his or her age. If we want to merge these two kinds of data, we need to transform numeric data values to discrete bins where necessary. For instance, when we want to use a LONGEVITY attribute, indicating how long the customer has existed in our database, we should discretize its values into bins:

\[
\text{LONGEVITY} = \begin{cases} 
\text{SHORT}, & \text{LONGEVITY} < 6 \\
\text{MEDIUM}, & 6 \leq \text{LONGEVITY} \leq 18 \\
\text{LONG}, & \text{LONGEVITY} > 18
\end{cases}
\]

### 3.3.3 Data preparation

As in the churn prediction workflow, the data is accessed through SQLQuery PEs, but in this scenario the data preparation phase requires more effort. First of all, we need to transform a relational data representation into a transactional representation:

![Figure 10: Transactional representation of the data](image)

In the transactional data representation, items purchased in the transaction are indicated with a Boolean true value, others with false. Then, if the data mining expert includes a customer’s data, the discretization has to be performed:
These two kinds of data are then joined into one data stream with a specified foreign key. The data can also be vertically or horizontally restricted with one or more TupleSelect or TupleProjection PE’s. Then, the stream is ready to be processed by an association rule mining algorithm.

3.3.4 Association rule mining

In the first version of the scenario, we used association rule mining algorithms provided by the WEKA library – Apriori and FPGrowth algorithms. Both algorithms cache the input data stream and perform data mining on it. This solution was not appropriate for large portions of data that was unable to fit into the memory. This problem led us to develop our own streaming implementations.

For the use-case purpose, we decided to use our own implementation of the Moment algorithm wrapped in the Massive Online Analysis (MOA [17]) framework. The algorithm is based on two tree data structures used together in processing [16]:

- Frequent Pattern Tree (FPTree) – stores content of a sliding window
• Closed Enumeration Tree (CET) – maintains only special kinds of item sets giving an overview of the whole processed data

In the current version of the scenario, we provide two similar PE’s for association rule mining:
BuildAssociator – standard PE wrapping WEKA algorithms (Apriori, FPGrowth). Produces output after all of the transactions are processed.
BuildIterativeAssociator – PE wrapping MOA association rule mining algorithms. Produces output every N transactions processed (where N can be specified by the data mining expert).

Figure 13: BuildIterativeAssociator PE
All inputs apart from outputFrequency are identical to BuildAssociator PE. The value specified as outputFrequency determines the number of tuples between associator outputs. Options for the Moment algorithm are also different when compared with the other algorithms:

- I – index of the present item indicator. Typically, data on input has two nominal values – true and false. Value of this option determines index of “true” value in nominal values list.
- S – minimum support threshold
- w – size of the sliding window

With this implementation, the workflow can be reused in various kinds of data mining applications requiring real-time data processing, where data arrives continuously in high speed, such as web click-streams or network traffic monitoring.

### 3.3.5 Association rule representation

The final step of the workflow is the rules post-processing. If many rules are being extracted, it is crucial to display them in a way appropriate for the domain expert. Apart from rule content, it is very important to provide a sorting facility based on the association rule quality measures described in 3.3.1.1. For this purpose, we use three post-processing PE’s:

- **AssociatorXML** – transforms associator to simple XML representation
- **AssociatorPMML** – transforms associator to Predictive Model Markup Language (PMML) representation
- **PMMLSVG** – transforms PMML representation to SVG visualization

--- CUT ---

```xml
<associationmodel minimumConfidence="0.1" minimumSupport="0.1" numberOfItems="4" numberOfItemsets="4" numberOfRules="12" numberOfTransactions="11999">
  <item id="0" value="web hosting & email box=t"/>
  <item id="1" value="Internet Access DSL=t"/>
  <item id="2" value="TV Digital=t"/>
  <item id="3" value="Static IP=t"/>
  <itemset id="0" numberOfItems="1" support="4221">
    <itemRef itemRef="0"/>
  </itemset>
  <itemset id="1" numberOfItems="1" support="5071">
    <itemRef itemRef="1"/>
  </itemset>
  <itemset id="2" numberOfItems="1" support="5044">
    <itemRef itemRef="2"/>
  </itemset>
  <itemset id="3" numberOfItems="1" support="4971">
    <itemRef itemRef="3"/>
  </itemset>
  <associationrule antecedent="0" confidence="0.3503909026297086" consequent="1" lift="0.82909495" support="1479"/>
  <associationrule antecedent="0" confidence="0.33996683250414594" consequent="2" lift="0.80873555" support="1435"/>
  <associationrule antecedent="2" confidence="0.33564631245043614" consequent="1" lift="0.79420626" support="1693"/>
</associationmodel>
--- CUT ---
4.4.7. Workflow description

Description of workflow for the cross-selling scenario is placed in Appendix A. The workflow contains three function and main workflow processing:

**PE(<Connection data> => <>)** deliverRules(String baseFileName, String hostname, Boolean passive)
produces three results – produce three results from the input data stream:

- XML representation
- PMML representation
- SVG visualization

Then, it call repeatedDelivery() function to store file with each result in the Repository. All files names start with string specified in baseFileName argument, but each file ends with different extension.

**PE(<Connection data> => <>)** repeatedDelivery(String fileName, String hostname, Boolean passive)
delivers data from input stream to the Repository to the file specified with fileName attribute.

**PE(<Connection data> => <>)** sampleResult(String condition)
delivers data from input stream to the client. The condition argument is SQL92 expression restricting scope of the stream. The function is used to produce data displayable in the ADMIRE Workbench visualisation tools.
4 Automatic Photometric Classification of Quasars

Quasars are highly energetic cores of galaxies, where matter is falling into black holes, releasing prodigious quantities of energy in the process. They are star-like in appearance and distinguishing quasars from stars requires information from the distribution of their light across the electromagnetic spectrum. The vast majority of star-like objects are stars rather than quasars.

Richards et al.[19] report success in using photometric methods to automatically classify quasars across five photometric bands taken from the Sloan Digital Sky Survey[20]. This pilot application investigated if the accuracy of classification could be improved by using 9 photometric bands by combining the SDSS data with UKIRT Infrared Deep Sky Survey (UKIDSS) [21] data.

4.1 The data

The data for this study comes from two astronomical surveys: SDSS and UKIDSS. Figure 15 shows an overview of the data in these databases. The SDSS database has over 585 million astronomical features. For each of these features it can provide five photometric band attributes known as u, g, r, i and z. Additionally SDSS also contains a table identifying 77,429 of these features as confirmed quasars. The UKIDSS database contains 36 million astronomical features. For each of these features it can provide four photometric band attributes known as Y, J, H and K. Additionally the UKIDSS database contains a link table that maps features in the UKIDSS database to nearby features in the SDSS database. This table can be used connect a feature in the UKIDSS database to the nearest feature in the SDSS database. For the purposes of this study combining the data from UKIDSS with the data for the nearest feature in SDSS is a valid join strategy.

![Figure 15: Data from the SDSS and UKIDSS data sources.](image-url)
After filtering out features that are not of interest to this study we are left with approximately 7 million features in SDSS and 8 million features in UKIDSS.

The astronomy community has developed a proposed standard called Table Access Protocol (TAP) [22] for accessing the astronomical databases. This is a web service based protocol that supports the execution of Astronomical Data Query Language (ADQL)[23] queries and can return data in several astronomical data formats including VOTable[24]. For the purposes of this study TAP was used as the interface to the astronomy data. This allows the use case to investigate how well ADMIRE’s architecture can support the inclusion of domain-specific interfaces such as TAP.

4.2 Data access and integration workflow

ADMIRE provides data intensive engineers with a variety of join strategies for linking data. For equi-joins, like the one required in this use case, the possible strategies include:

- In-memory join: where input to the join operation is read into memory and efficiently indexed before the other input is streamed through.
- Ordered stream join: where both inputs to the join operation are ordered by the value of the join attributes.
- Batch queries join: where values from one input are used to send multiple queries to produce the data for the other input in batches.

Given the number of features involved it is not feasible to perform an in-memory join. This is precisely why ADMIRE is designed around a pipelining model that prefers processing elements that have a low memory footprint.

The ordered stream join is a possibility, but unfortunately the SDSS database schema does not support the efficient extraction of the data ordered by \textit{sdss}\_\textit{id} attribute obtained from the link table.

![Diagram of Quasar detection batch query join strategy.](image)

Figure 16: Quasar detection batch query join strategy.
The batch queries join strategy is a good fit for this use case. Data can be streamed from the UKIDSS database without any ordering constraints and then for each \( n \) tuples from UKIDSS a query can be sent to SDSS to extract the data for the \( n \) features identified by the \( sdss_id \) attribute from the UKIDSS data stream. This strategy can be seen in Figure 16.

The batch queries join algorithm scales well and can support very large data sets, but it can be slow as it requires many queries to be sent to the second data source. It is important to choose a batch size that optimises the execution time. While it would probably have been more efficient to use a higher value, a batch size of 250 was used. This value was chosen because higher values led to very verbose queries that the TAP server implementation was unable to cope with.

The execution time of the batch query join strategy can sometimes be significantly reduced by executing multiple parallel queries to the second data source. This proved very useful, with ten parallel branches reducing the time of execution to about 26% of that of a single branch. The speedup gained through parallel branches depends very much on the ability of the data source to scale to multiple queries. In this instance the parallel branches are all querying the same single machine TAP server which is querying the same back end database.

A further complication arose at implementation time when it was discovered that the UKIDSS TAP service would not stream the result data from the initial query. The same query when sent directly to the database successfully streamed the result, but the TAP server would transform the query in such a way that the backend database was no longer able to stream the result data. To overcome this problem the UKIDSS database is also queried in batches. Each query selects only those features that lie in a unique subset of a sky. By adopting a strategy of many smaller queries that return quickly we are able to produce a chunked data stream that allows the rest of the workflow to being processing whilst the UKIDSS database is still producing data. A simplification of the whole data integration workflow can be seen in Figure 17.

The workflow in Figure 17 makes use of several key features of DISPEL and the ADMIRE infrastructure:

- A programmable PE is used to provide functionality not provided by any existing PEs. It is used to produce tuples streams that output the consecutive ranges of the sky to use in each query to the UKIDSS data source.
- The for loop functionality of DISPEL, which makes it easy to implement and adjust the level of parallelism.
- A core set of basic PEs that provide the pieces required to implement the batch queries join algorithm.

### 4.3 Performing the data mining

Having combined the data into the required format, the next task is to generate classifiers to analyse whether using nine photometric bands attributes produces a better classifier than using only five. It would be possible to store the data set for subsequent data mining using a \( k \)-fold validation technique, but instead this use case looked to investigate how the ADMIRE infrastructure could be used to produce an initial result very quickly in order to identify whether a larger scale analysis was likely to yield useful results. The ADMIRE infrastructure proved to be ideal for this, as it easily supported implementing the following tricks in DISPEL:

- Reducing the size of the data set by restricting the queries to only cover a subset of the sky
- Dividing the data sets into training and testing sets where the first \( n \) tuples form the training set and the all subsequent tuples form the testing set. Splitting the data in this way means that the testing data set does not need to be buffered while training is taking place. Such tricks are ideal for a pipelined architecture but a domain expert must make a judgement as to whether it is likely to produce a bias in the data.
- Performing the training and classification of both the five attribute model and the nine attribute model in parallel and displaying the results for each at the same time so they are easily comparable.
The full DISPEL document for this use case can be seen in Appendix B.

Figure 17: Quasar detection workflow.
4.4 Conclusion

The ADMIRE infrastructure has been ideal for exploring this use case. Initially, PEs had to be developed to query the TAP servers, but once this was completed it was easy to integrate these services into ADMIRE workflows. DISPEL provides an easy way to construct these workflows and the multi-threaded, pipelined nature of workflows makes it very easy to implement multiple parallel branches for the batch query execution.

Parts of the workflows used here could be encapsulated in common functions or generic PEs for easy use by others. For example, the batch join pattern is a very common distributed data integration pattern and could be supported by a DISPEL function that implements the framework of the pattern. Another example is the invocations of the programmable PE to add list markers and generate tuples that specify ranges which could be wrapped as generic PEs and added to the ADMIRE repository.

This approach has proved successful in supporting a quick investigation of ideas. As the core set of ADMIRE PEs increases it will become easier and quicker to construct simple workflows that extract, merge and mine data sets to see quickly see if there is merit in an idea.
5 Seismic Noise Interferometry

Seismic noise interferometry focuses on the ambient noise signatures originating at a distance from the surface receivers, in order to retrieve the Green function between these receivers. It utilizes the cross-correlation of signal pairs to reconstruct the impulse response of a given media.

The processing steps needed to perform such calculations are pretty much defined in literature, although the software adopted is often extremely customised and the data needs typically be stored locally to the place where the calculation takes place. Our experiment aims to the development and deployment of an infrastructure, as a proof of concept of the actual possibility of interconnecting, through the ADMIRE framework, a set of distributed and independent European data centers.

At the same time, aspects related to the modularisation of the scientific software, provenance management and the adoption of community driven standards have been also taken into account, demonstrating the advantages and highlighting the limitations of the current implementation.

Therefore, the scope of this section will be mainly focusing on the technical and architectural solutions adopted, rather than on the value of the scientific results obtained. The tuning and the improvements needed to accomplish the scientific task will be eventually obtained with the engagement of the domain-experts, providing the appropriate parameters and the additional analytical software that the deployed architecture will be able to ingest and execute on the actual data.

5.1 Topology of the infrastructure

The current seismology Testbed exploits two different geographically distributed data providers represented by significant and well established European data centres and institutions: ORFEUS and the British Geological Survey (BGS).

As a matter of fact one of the main aims pursued by this use-case was to test the behaviour of the ADMIRE infrastructure in dealing with heterogeneous data and metadata but also with diversely geolocated sources. This scenario is very well depicted here where apart from embracing a common standard for the seismological raw data called MSEED, each institution has its own way to store and manage data and metadata, thus requiring a certain grade of flexibility and adaptation. In our example we have two different file systems holding the raw data whereas for the station metadata (gain,
response, frequency, etc) we utilize a well-known ASCII format (RESP files) stored at BGS and ORFEUS.

An important issue to take into account in a data intensive platform is the minimization of data transfers and shipments throughout the system, thanks to the ADMIRE framework we were able to tackle this problem effectively by moving some of the processing steps close to the data.

### 5.2 Description of the workflow

Ambient noise processing is usually performed in time windows on a predefined set of seismic stations. These can be considered as the main inputs together with a number of domain specific parameters and can be provided to the system via portals or other types of interfaces, ultimately through a DISPEL document. In this context the choice of the optimal time window length is relevant because it could affect the final result of the analysis. Ideally it is desirable, from a user point of view, to being able to specify as the duration of the overall period of interest as the single window length. In order to address these requirements we introduced a TimeSampler as the first element in our workflow. This PE behave somehow as a clock for the system, chopping the overall interval of analysis (in general months or years) up into sub intervals (typically hours, days) giving the user the possibility to configure these parameters and eventually allowing comparisons of intermediate results and dynamic adjustments.

In output of the sampling phase there is a list of time windows \([T_1,...,T_n]\), therefore from now on we can refer to the single time \(T_i\). Subsequently a query, composed by a list of stations \([S_1,...,S_n]\) plus the time \(T_i\), is sent in parallel to several Data and Metadata Extraction modules. The main task of these modules is to interface the datasources providing a direct access layer and preparing a uniform output for the next steps. As already mentioned the great challenge to tackle at this stage was the heterogeneity in the data systems.

We had to cope with different data and metadata storage and management mechanisms such as legacy archives. For this purpose ad hoc OGSA-DAI activities have been written and then mapped into a generic PE (WFRetrieve) whose specific implementation depends on the particular gateway where it is deployed to, thus providing a unique transparent interface. Once data and metadata are fetched from the archives, they can be streamed over the next blocks.
In order to minimize data shipments we moved some preprocessing steps like filtering, normalization, instrument response removal, whitening and decimation close to the data itself, in fact these operations are typically serialised into pipelines that can be parallelised, exploiting the resources available at the data center location, contributing eventually in reducing the data flow. For each operation, the activities have been implemented trying to make large reuse of the already existing domain specific libraries, as we will describe later in this section.

The WFExtractionPE is a composite PE made of two main sub elements, WFRetrieve and an AppendAndSyncPE which is responsible for shaping the data stream based on the specific requested time window. The preprocessing steps are then assembled in a pipeline structure where at each step the stream is treated accordingly; this approach provides great flexibility and allows us to extend, change, re-shuffle the elaboration by just rearranging the pipeline or plugging in new PEs. Appendix B shows how this has been implemented using DISPEL.

At the end of the pipelines we obtain a collection of preprocessed streams which are almost ready for the cross-correlation, however some effort is still required to merge the streams together choosing the most appropriate ones. Because each data centre is completely independent from the other, they could be for example providing data regarding the same station. Therefore we had to take in account the existence of duplicates and set up mechanisms for an optimal choice based on some quality parameters. In our case we chose to use the total Number of Gaps (NoG) present in the traces as discriminant factor, which could seem a quite rough measure not considering the duration of these gaps. Nevertheless we assumed that the data are pre checked in advance by the data centers.

A merger PE, StreamHarvester, has been implemented to accomplish this task: after selecting the best \( n \) streams matching the initial query, they can be pushed further and eventually maintained into an intermediate storage systems useful for recoveries and successive re-elaborations of the workflow. This feature coupled with the Data Provenance mechanism discussed in a next paragraph contributes to insure reliability and repeatability of the overall process, improving the quality of the scientific
results. We believe that in order to set up a robust and stable automated data processing procedure it is fundamental to adopt proper data quality control measures, the traceability of the experiment together with the monitoring can be for example a good indicator in the validation and verification steps.

We are ready now for the primary objective of our analysis: pair wise cross correlation of the pre elaborated time series.

Figure 21: Asymmetric cross correlator.

![Asymmetric cross correlator](image)

Figure 22: Cross-correlation using different shift sizes for time windows (in seconds)
(a) tshift = 500s (b) tshift = 250s

![Cross-correlation](image)

Here the effort is mainly focused on expressing the functionality descriptively, without any knowledge of the underlying infrastructure. How this feature is then enacted depends on the specific deployment. We expect the ADMIRE framework to take care of the optimization considering the appropriate level of parallelism. This constitutes a fundamental and necessary step to upgrade to a production level quality since the cross correlation of a large number of time series over long period can be a very time and resource consuming operation. So far we have been appreciating how the separation of concerns, one of the pillars of the ADMIRE framework, provides a powerful approach to the solution of complex scientific problem, but on the other hand we think that the lack of a proper optimization layer could represent a strong barrier to the real world.
The final results are presented via images showing the correlation functions between pairs of stations. Therefore there is an immediate visual feedback for the users that can be very helpful in adjusting and tuning the experiments while still running.

5.3 User-Defined Processing Elements

The building blocks of a DISPEL workflow are the PEs, the execution of which is optimised and parallelised throughout the nodes of the ADMIRE computational infrastructure, which relies on the OGSA-DAI data-streams processing technology. An important effort has been the dedicated integration of the domain specific API ObsPy [59] into the underlying OGSA-DAI activity framework, as the low level implementation for the seismological analysis PEs. This implementation choice was driven by a more general and important requirement of the seismology scientists who expect to develop and use their own analysis code, written with their favourite programming language and API.

Besides the investigation on how to realise such integration, a more general requirement suggested we implement a small framework (Figure 23) to abstract away those aspects related to the most basic and important needs of this genre of activities. These aspects concern: the configuration of the activity that has to execute a specific analysis code within a particular runtime environment (Python, Fortran, C...), the passage of the parameters and the data between these layers, the production of the metadata in an interoperable format and the deallocation of memory resources.

All the yellow classes in the UML representation of Figure 23 are part of the framework. The SeismoActivity and the SeismoResourceActivity, which inherit from a core OGSA-DAI activity class and interface, are the activity classes that implement the processing PEs. A SeismoActivity accepts one INPUT and a set of PARAMETERS. Once they are read, they are passed to an userDefinedProcess that takes care of organising these information before passing them to an implementation of the IExecutionWrapper, which is responsible for running the script performing the actual transformation on the data. Once the transformation is applied the result is returned back to the wrapper and eventually to the SeismoActivity where it gets written back into the dedicated connections.

Sometimes a specific task required us to extend the SeismoActivity. This might happen when the input is not a datastream but, for instance, a list of file references. As an example, the AppendAndSyncActivity reads a list of MSEED files from a filesystem datasource, then it merges and
slices the time series contained into the files according to a specific time window passed as a parameter. Eventually the activity converts the time series obtained into a datastream to be delivered through the OUTPUTDATA connection. Notice that the AppendAndSyncActivity extends from SeismoResourceActivity, which is already providing to its subclasses all those methods that enable the access to a specific resource, a file system in this case.

As previously mentioned, we had to deal with the execution of domain specific analysis code written with the Python library ObsPy[25]. Therefore the implementation of the PythonWrapper represented an extremely important task in order to enable the communication of OGSA-DAI with the underlying Python technology. After investigating the spectrum of the solutions already available to help such integration, we decided to implement the wrapper adopting JEPP [28], a library designed to embed CPython in Java within a heavily threaded environment, which is exactly what we have to achieve in our use case.

Another detail that is worth mentioning is that the actual data transferred between the analysis PEs is a serialisation of the ObsPy Stream object representing the data, rather than the underlying data stream in MSEED format. The choice of transferring the Stream object, serialising/deserialising it within the Python code, has been somewhat forced by the lack of a feature in ObsPy that allows the acquisition of MSEED standard volumes from a memory buffer. Currently, the library allows one to read this domain specific format only by passing a reference to a MSEED file, thus breaking the streaming model, a strong limitation for a proper embrace of the ADMIRE philosophy.

The approach just described to overcome such limitation obviously binds the content of the data stream to the specific technology which is able to deal with it, requiring, for the sake of interoperability, the development of converter PEs to transform the Python object either into a MSEED file or into a MSEED stream, and vice versa. In order to have more flexible and interoperable solution, the ObsPy community has been solicited in considering this aspect in the further developments of the seismology API.

To conclude, the small framework just presented provided us the ability to assign a behaviour to a PE realised as a SeismoActivity, directly within the ADMIRE configuration system. The adoption of the Spring technology [26] allowed us to configure the properties of such PEs just by editing an xml file, declaring a different configuration for each PE name, specifying which IExecutorWrapper to use and which analysis script to execute. In other words, different PEs names can be bind to the SeismoActivity class, and the behaviour is determined within the SeismoActivity at runtime by accessing the Spring context, thus obtaining the desired configuration depending on the name of the PE invoked.

### 5.4 Data Provenance

The increasing amount of data needed to perform the type of analysis required by this use case, lead us to acknowledge the existence of all sort of information about the creation process of the data streams that contribute to the final result, which is obtained eventually by the aggregation and stacking of a large quantity of intermediate and partial results.

For the user of a workflow framework like ADMIRE where several and distributed services are used to accomplish complex computational tasks, it would be interesting to understand how the result from such a computation was created and on which node. Therefore, storing and accessing provenance information can be used to examine the derivation process, the data quality at a certain point of the execution of an experiment or to re-run the experiment itself, either on the partial results already available or on a subset of the datasets that have been already processed and stored into intermediate archives.

In [27] we can find the description of the two main strategies to approach the recording of provenance data: The Eager and Lazy approach. While the first is often based on the collection of annotations at each step of the processing, carrying them along till the end of the computation, the latter suggests that the processing of the provenance should happen only when needed. It assumes also that the provenance information of a certain result can be inferred by running a query on a database that is obtained as a transformation of a source database performed by the execution of the computational task.

---

**Seismic Noise Interferometry**

39
Our choice for collecting provenance largely embraces the lazy approach, leaving the workflow developer to choose when in the computational process is relevant to store the provenance metadata. Such knowledge is directly produced from the single analysis PE that performs a transformation on the data stream. When needed this information is extracted by a set of dedicated Provenance PEs, processed and stored into a centralised relational database. In case of errors, these are also extracted from the analysis PEs and stored into the relational model, as for the positive executions. The Provenance PEs can be connected together in order to keep track of the story of the metadata produced from the analysis PEs to which they are attached to (Figure 24). At the end of the computation, a query Q on the relational database schema can provide all the relevant provenance information that lead to a specific output result.

As we have mentioned, ADMIRE is a framework that enables the development of workflows which consist of composed Processing Elements that are meant to be executed often as close to the data as possible, within a distributed infrastructure. This suggests that retaining the knowledge about where the computation is happening can become extremely useful, for example, to detect and eventually address the weaknesses of the workflow, that are sometimes related either to the deployment of the infrastructure or to the quality of the data provided by one of the nodes feeding the stream. To achieve this we include within the dataset’s metadata a Request Identification which is generated by the Gateway itself and assigned to the subgraph of the workflow that has been executed. This will allow us to reconstruct the topology of each run of the workflow enabling, for example, to evaluate how a processing subgraph executed within a specific Gateway is contributing the overall performance or to any failure that might occur.

The ER schema adopted to represent the Xcorrelation process (Figure 24) assumes that during the preprocessing, a sequence of dataset’s metadata is produced, as the result of the transformation applied on the stream by the analysis PEs, which is invoked with a set of parameters. The datasets are generally composed of one or more traces (time series) and each transformation on these time series produces a new dataset with a new set of traces, the transformed ones. The Xcorrelation relationship combines together two traces producing the intermediate results before passing them to the stacking phase of the workflow, which lead to the final result. Thus the Xcorrelation relationship retain the
information needed to start tracking back all the traces adopted, the history of the transformations applied on them and the gateways performing those.

Storing the errors might also become important at this point; for example, if the final result will present less stations than the one specified by the initial query. In such a case tracking the RequestID of the subgraphs involved into the computation could provide the information needed to understand at what point of the workflow the missing stations disappeared and why.

The need to update our schema consistently with the foreign key constraint that have been specified among the tables, revealed a missing feature of the ADMIRE framework: it does not currently provide a generic activity to execute a list of SQL update/insert expressions on different tables subsequently, in order to respect such constraints. Our solution to this task is to embed within the set of Provenance PEs, the logic that performs the updates on the database sequentially. This approach helped also for the creation of the primary keys, that are obtained into the virtual machine of the gateway hosting the provenance database, avoiding in such a way the risk of creating keys that are not unique.
6 Automatic Gene Annotation of Mice Embryos

The application described in this chapter comes from the EURExpress-II project\(^7\), which aims to build a transcriptome-wide atlas of gene expression for the developing mouse embryo (illustrated in Figure 25) established by RNA in situ hybridisation. The project annotates images of the mouse embryos by tagging images with terms from the ontology for mouse anatomy development. So far, people analysing the images have manually done all the annotations. ADMIRE aims to automate the whole process.

To understand the process one needs to understand the data first. There is 18,000 genes’ collection for mouse embryo established by RNA in-situ hybridisation. The data consists of mouse embryo image files and an annotation database that describes the images. There are around 4 Terabytes of images out of which 80% have been already annotated. This leaves 20% of them for ADMIRE, which is still over 85,000 images. Each image can be tagged with terms from the ontology set of 1,500 anatomical components. Corresponding tags are stored in the annotation database.

6.1 Overview of the annotation process

In order to automatically annotate an image there are several steps that need to be taken in the process as illustrated in Figure 26:

- Data Integration – data comes from different sources: annotations are stored in a databases and images on a disk.
- Image Processing – there are several operations that can be applied to the images, most important being: size scaling and noise filtering.
- Feature Generation – we need to generate features representing different gene expressions patterns which we can than analyse; we do it by applying a wavelet transform.
- Feature Selection/Extraction – there is a large number of features and they need to be reduced by selecting and/or extracting the most important features; we apply Fisher Ratio Analysis.
- Classifier Design – we build separate classifiers for each of the anatomical components.
- Testing and Evaluation – by taking an image (previously annotated) and applying to it the classifier we created in the previous step we want to predict if an anatomical component is

\(^7\) EURExpress-II, http://www.eurexpress.org/ee/
expressed in this image or not. We can than compare the actual result to the predicted one and rate the classifier.

- System Deployment and Automatic Annotation – once we built classifiers that have high success rate we deploy them on the system; then we take new, previously not annotated images and try by applying the classifiers we automatically annotate them.

![Diagram of process](image)

Figure 26: The process of enabling automatic annotations for EURExpress-II.

### 6.2 Evolution of the use case

The EURExpress-II use case was implemented in ADMIRE at an earlier stage of the project. It was at the time when the framework was still evolving and it helped to shape the architecture. Now, having the framework more mature, we decided to have another look at the use case and implement it again. We planned to tackle it on two fronts. Firstly, by analysing what we learned about the use case doing it the first time round. Secondly, we wanted to take advantage of the latest features of the framework, especially the DISPEL language and apply them to the version two.

### 6.3 New approach

In the first version of the use case, we had one DISPEL document describing all the steps of the annotation process. We noticed that several steps are done only once. Therefore, executing them every time was unnecessary. We decided to split the whole process into two stages: the data preparation and the data mining stage.

In the first stage, we are concerned about integrating and pre-processing the images and generating the features representing gene expression patterns. This stage needs to run only once (for all concerned images) and the results are stored and can be reused in the later stages many times.

In the data mining stage, we select the most important features and based on them we train the classifiers. Having done it, we can test and evaluate how good the classifiers are. Once we have a classifier, which predicts sufficiently well, we store it for later automatic annotation of a particular anatomical component in new images.

### 6.4 New features

There are several new features of ADMIRE which were not available at the time of implementing the first version of the use case.
First of them is the existence of the ADMIRE Repository. We use it for storing the features generated from the images and the indices of selected features. This lets us split the execution and repeat parts of it if necessary. It can be also used for storing trained classifiers.

As the DISPEL language has progressed significantly since we implemented this use case for the first time, we can now use many of its new features. Arguably, the most important addition is the use of functions and abstract and composite processing elements. We refactored the code and build it around functions, so it can be easily reusable. We factored out all the important parameters and set them as input arguments to the functions, so the same experiments can be run with different parameters with a minimal effort. Figure 27 illustrates the data preparation stage of the use case. All the processing elements forming the part of the composite PE returned by the function are presented in the grey rectangle. Boxes placed on the white background with arrows pointing inside of the grey area are function arguments.

Figure 27: Visualisation of the data preparation stage DISPEL document.
An additional benefit of having functions is the possibility of recognising and supporting patterns. In the data mining environment a very common pattern is k-fold cross-validation. As it is already implemented in DISPEL we can use it "off the shelf".

Another new feature of DISPEL syntax is “repeat enough of”. It significantly simplifies the DISPEL document (in terms of creating and reading) and there is no necessity of manual insertion of ControlledRepeat blocks which are added automatically by the ADMIRE Gateway. There are two “repeat enough of” blocks shown inside of the composite PE in the Figure 27.
7 Application Portals in ADMIRE

ADMIRE application portals are user-friendly, domain expert-oriented web portals that enable high level access to ADMIRE infrastructure wrapped in a domain specific user interface. What that means is that the whole ADMIRE infrastructure is hidden behind the user interface and the user interacts with the portal as with any other domain focused application. Behind the scenes, the portal uses the ADMIRE tools and services to perform requested tasks. In particular, it constructs DISPEL documents based on parameters provided by the user, submits them to the Gateway for enactment and displays the results. When everything works smoothly the ADMIRE software layer is completely transparent.

In following sections we first touch on the underlying portal technology (for more details see D5.6 [18]) and then introduce individual application portals.

7.1 Architecture and Technology of ADMIRE Application Portals

Application portals in developed in the ADMIRE project are based on the Google Web Toolkit\(^8\) (GWT). GWT based portals provide better, more dynamic user experiences thanks to the employment of Javascript and the AJAX architectural model, yielding a more responsive user experience. From the portal developer’s point of view, GWT allows faster development compared to older portal technologies through its use of the Java language for both server-side and client-side (i.e. browser) programming, the existence of a lot of predefined components, and tight integration with leading Java development tools. GWT shields developers, and therefore also portal users, from many cross-browser compatibility issues, again reducing development time.

![Interaction of Experts with ADMIRE Framework and Portals](http://gwt.google.com)

ADMIRE portals were developed as domain specific user interfaces, meaning that there is no generic portal application for the overall project. However, all ADMIRE application portals use common client libraries for the ADMIRE services like gateways, registry or repository.

\(^8\) http://gwt.google.com
The place of the application portals in the ADMIRE tool-space is shown in Figure 28. It uses predefined DISPEL files, prepared e.g. by data mining experts using the Workbench, to execute domain specific workflows on the Gateway and display their results.

7.2 ACRM Churn

The ACRM Churn Prediction Portal provides a domain oriented interface for ACRM analysts. There are two general activities that can be performed in this Portal:

- Browsing predictive models produced in data mining workflows
- Detecting churned customers with models

Each predictive model stored in the Repository can be displayed as a text file. Typically, it is a decision tree representation. The Portal does not have to use a fixed Repository address.

The Portal uses a built-in DISPEL workflow to perform detection of churn customers. The domain expert has to specify only models (Figure 29) and the input data to be used for prediction (Figure 32). The data can be stored in a CSV file or can be typed in manually. After these two parameters are specified, the Portal application replaces the appropriate values in the DISPEL files and executes it on the Gateway (Figure 30).
Typically, this process is very quick, because it does not require data-intensive processing. As soon as this process is finished, results can be displayed in the table of customers data from the input by selecting the “Mark churned customers” button. Rows with customers detected as churned are marked in red colour.
7.3 ACRM Cross-selling

The Cross-selling Portal is made with the same technology as the Churn Prediction Portal. The difference is that there is no DISPEL execution from the Portal side. The Cross-selling Portal aims to provide a domain expert with an easy way of browsing association rules. This Portal can display association rules delivered to the Repository in real-time. All details of rules can be displayed and, crucially, also sorted. There are three ways to display a rule set in the Portal:

- Expandable list (Figure 32)
- Visualisation (Figure 33)
- Plain PMML

The picture below presents an association rule set displayed in the most useful way – the list:

Figure 32: Cross-selling Portal displays association rule set
7.4 FFSC ORAVA

The Flood Application portal for ORAVA scenario aims to provide hydrology experts with domain oriented user interface for researching hydrological data from the Orava region in Slovakia. The graphical user interface (GUI), shown in Figure 34, exploits data mining and integration workflows developed and pre-configured by data mining experts in the context of the ADMIRE project.

Figure 34: ORAVA scenario user interface.
The GUI allows the user to perform data integration, model training, model validation and forecasting. Forecasting and validation differ in input data provided to the trained model. Validation uses historical data, while forecasting uses current data. Data integration requires the user to choose one gauge station from a list for which the data will be extracted and integrated with other environmental data like air temperature, rainfall, and discharge from upstream reservoir and so on. The integration is done for user specified time period.

Once there is an integrated dataset, it can be used for model training, validation or prediction. Training phase requires the user to specify which integrated data set to use and produces trained model, which can then be validated using another integrated data set or can be used for forecasting when current data is provided as input.

All the operations just mentioned use common job execution and management infrastructure provided by the ADMIRE project. Every operation is expressed as a sequence of `dispel` statements, which are submitted to ADMIRE Gateway in one block. Such submission is referred to as a gateway process or job.

The jobs are listed on a dedicated page of the application portal (see Figure 35). Each job has an internal ID, a name, status, execution time and a link pointing to results or error description in case the job execution fails. Status codes are the most important information and are therefore distinguished by color.

Figure 35: List of jobs submitted by user.
Once the job has finished execution – signaled by a COMPLETED status – the user can browse the result using provided link, which switches to a Data View portal page (see Figure 36). This page displays the data in two forms: as a browsable table and a zoomable chart.

If the job fails, an error description link points to a text box containing error message, which falls into one of the four categories: compile error, registration error, runtime error or system error. The level of detail of individual error message depends on the subsystem that produces it.

Finally, in order to support more advanced users, the portal offers the option to submit any DISPEL sentence through a Generic Job portal page (shown in Figure 37). Once submitted, such a job is added to the list of jobs on the Jobs portal page.
7.5 FFSC RADAR

The Flood Application portal for RADAR scenario aims to provide the experts with simple interface for executing precipitation forecast workflow based on computation of future meteorological radar data using motion vectors (see Section 2.3 for more information on the application).

The basic structure of the portal is similar to ORAVA scenario and allows the user to execute three workflows: training of the precipitation prediction model, actual prediction and computation of radar images only (see Figure 38). After the job is submitted, it appears on a dedicated page of the application portal (see Figure 35). Once it has finished, results can be displayed by clicking the “result” link in the row with job’s status information (see Figure 39 for example).
The Flood Application portal for SVP scenario aims to provide a simple interface for data integration, model training and actual forecast of water flow into Orava water reservoir at the beginning of the Orava river.
The structure of user interface is very similar to ORAVA scenario, but there is no choice of stations, because we deal with one concrete water reservoir. We also use three separate steps: data integration, model training and prediction. However, the data sets are completely different (see Section 2.4 for more details on the application).

Figure 40: SVP scenario user interface.
8 ADMIRE Platform Evaluation

In previous chapters several application use cases have been presented that have been developed using the ADMIRE infrastructure and tooling. The individual use cases already presented some thoughts on the usefulness of the infrastructure and how they used it. In this final section, we try to assimilate and express the lessons learned and provide an application-oriented evaluation of the experimental ADMIRE platform. An in-depth evaluation of the ADMIRE architecture using results from the use-case experiments is presented in D2.9 [29].

We begin with an observation that developers of the use cases have taken advantage of the various tools provided to create working applications. They have used the Workbench to construct, test and debug DISPEL documents to implement the workflow of their applications, the Repository to store the results of their data processing workflows and retrieve data for further processing, and the Registry to register existing and newly created PEs to make them available to all the tools. They have used portals to create high level domain specific user interfaces that hide the low level ADMIRE infrastructure in order to allow the experts to focus on their problem domain.

The core of the application development using ADMIRE infrastructure and tools lies in the creation of the DISPEL documents. The DISPEL language and distributed network of Gateways allow developers to create complex workflows that are easier to execute and change than before. The DISPEL language allows developers to describe the processes at a high level of abstraction, independently of any low-level concerns regarding the underlying enactment engines, databases or any consideration of the distributed environment. The use cases have made use of interconnected gateways, which together provide the necessary data, processing elements and visualization tools which the use cases require. The addition of functions, abstract and composite processing elements to the DISPEL language have been exploited by several use cases, because they provide important modularization and code reuse capabilities.

This novel approach allows the infrastructure to be easily extended with new data providers, by deploying a gateway at the site of the new provider, and registering it with the other gateways. Then, when a data analysis expert creates a DISPEL document that makes use of one of the capabilities provided by this gateway, it can be accessed and integrated automatically into the overall knowledge discovery workflow. Additionally, this approach allows remote high-performance data mining tools to be used, or other data storage facilities to be accessed.

This model provides a clear separation of responsibilities between data-intensive engineers, data analysis experts, and the domain experts of the application. The underlying infrastructure and gateway network is managed by the data-intensive engineers. The data analysis experts speak DISPEL and create full knowledge discovery workflows which use the infrastructure and in turn these workflows are used by the domain experts via specialized domain-specific user interfaces such as portals. Apart from separating the concerns of the involved stakeholders, this approach also separates the technology into fairly independent layers, and so a once-tuned DISPEL document will work even when the infrastructure changes significantly, provided that infrastructure is still capable of providing all of the processing elements referenced by the document.

This approach also provides for a reasonable amount of fault tolerance. If one data centre becomes unavailable, it may conceivably be replaced transparently by a different one, without bothering the users of the application. Some centers and gateways are, of course, irreplaceable in a given network (primary data storage centers for instance), but data filters may be deployed at several locations to enhance redundancy. There may be also several HPC facilities available to a given user, so the temporary inaccessibility of one of them is no issue – the DISPEL description of the required data-oriented solution is entirely agnostic of such things.

When developing new applications, new PEs might need to be implemented, deployed and configured in some of the gateways in order to use application specific data processing algorithms, which may present a considerable effort for developers, especially if they have no prior experience with ADMIRE infrastructure. However, once the PEs are in place the application development proceeds to higher level of abstraction where one needs to “just” connect the right boxes in correct order with the right
parameters. This experience leads to the realization of importance of a comprehensive library of PEs that will span different application domains. While multiple important PEs already exist, additional effort needs to be put into broadening the range of functionality covered, because we think that the library would play an important role in the uptake of ADMIRE infrastructure and tools. It is mentioned in several of our use cases that once the right PEs were in place, the focus could be put on researching the data by exploiting the capabilities of the ADMIRE infrastructure.
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACRM</td>
<td>Analytical Customer Relationship Management</td>
</tr>
<tr>
<td>APE</td>
<td>Application Processing Element</td>
</tr>
<tr>
<td>CRM</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>DGT</td>
<td>Data Generation Tool</td>
</tr>
<tr>
<td>FFSC</td>
<td>Flood Forecasting Simulation Cascade</td>
</tr>
<tr>
<td>LFN</td>
<td>Logical File Name</td>
</tr>
<tr>
<td>MARS</td>
<td>Monitoring Agriculture with Remote Sensing</td>
</tr>
<tr>
<td>OGSA-DAI</td>
<td>Open Grid Services Architecture – Data Access and Integration</td>
</tr>
<tr>
<td>OR&amp;A</td>
<td>Ocean Reports &amp; Analysis</td>
</tr>
<tr>
<td>PE</td>
<td>Processing element</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-oriented Architecture</td>
</tr>
<tr>
<td>UFS</td>
<td>User Friendly Schema</td>
</tr>
<tr>
<td>UISAV</td>
<td>Institute of Informatics of the Slovak Academy of Sciences</td>
</tr>
</tbody>
</table>
10 References

[16] Yun Chi, Haixun Wang, Philip S. Yu, Richard R. Muntz – Catch the moment – maintaining closed frequent itemsets over a data stream sliding indo
Appendix A  Description of Datasets And Custom Processing Elements

A.1  Data Used in Flood Forecasting Simulation Cascade Scenarios

A.1.1  Domain Data

This section first shows a summary of the data available for the FFSC application. Most of the data have been already described in [1] and [2]; in this chapter, we describe only data relevant to current application scenarios.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Domain</th>
<th>Description</th>
<th>Volume</th>
<th>Temporal coverage</th>
<th>Spatial coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARS</td>
<td>Meteorology</td>
<td>Historical meteorological data (temperature, rainfall, etc) for Slovakia</td>
<td>100s of MB</td>
<td>1975-2007</td>
<td>Slovakia (grid 50x50 km)</td>
</tr>
<tr>
<td>SVP</td>
<td>Hydrology</td>
<td>Data from waterworks in western Slovakia (mainly river Váh) – outflows, water levels, temperature, rainfall</td>
<td>100s of MB</td>
<td>1998-2007</td>
<td>15 distinct waterworks</td>
</tr>
<tr>
<td>SHMU_CURR</td>
<td>Meteorology</td>
<td>On-line database of meteorological data – copied from SHMI web; including radar imagery</td>
<td>10s of GB</td>
<td>2008-</td>
<td>Slovakia (about 100 distinct probes)</td>
</tr>
<tr>
<td>SHMU_HIST</td>
<td>Meteorology</td>
<td>Historical meteorological data from SHMI probes</td>
<td>100s of MB</td>
<td>1998-2007</td>
<td>Slovakia (more than 100 distinct probes)</td>
</tr>
<tr>
<td>SHMU_GRIB</td>
<td>Meteorology</td>
<td>Historical temperatures and rainfall amounts in a gridded binary format</td>
<td>100s of GB</td>
<td>1998-2007</td>
<td>Slovakia (grid, various sizes)</td>
</tr>
<tr>
<td>RADAR</td>
<td>Meteorology</td>
<td>Weather radar imagery</td>
<td>100s of GB</td>
<td>2005-2008</td>
<td>Slovakia (gird, homogeneous)</td>
</tr>
<tr>
<td>SHMU_HYDRO</td>
<td>Hydrology</td>
<td>Historical data from hydrological measurement stations</td>
<td>10s of MB</td>
<td>1998-2007</td>
<td>Orava and upper Vah rivers</td>
</tr>
<tr>
<td>SNOW</td>
<td>Hydrology</td>
<td>Amount of water retained in snow cover in the watershed area of selected reservoirs</td>
<td>10s of MB</td>
<td>1980-2009</td>
<td>Selected reservoirs in the Vah river watershed area</td>
</tr>
<tr>
<td>INFLOW</td>
<td>Hydrology</td>
<td>The inflow of selected water reservoirs</td>
<td>100s of MB</td>
<td>1997-2009</td>
<td>Selected reservoirs in the Vah river watershed area</td>
</tr>
</tbody>
</table>

Table 13: Summary of data sets used

A.1.2  MARS

A.1.2.1  Data Provenance

This data are freely available from WWW, and may be used in any manner, including for scientific purposes.
A.1.2.2 License

The data may be used without any restrictions.

A.1.2.3 Data structure

The Mars Stat Data Base contains meteo interpolated data from 1975, covering the EU member states, the new Independent states and the Mediterranean countries. The data set available in the ADMIRE testbed contains data for area of Slovak republic from 1975-01-01 to 2006-12-31 in a 50x50 km grid.

Available values are the following:
- maximum temperature (°C)
- minimum temperature (°C)
- mean daily vapour pressure (hPa)
- mean daily windspeed at 10m (m/s)
- mean daily rainfall (mm)
- Penman potential evaporation from a free water surface (mm/day)
- Penman potential evaporation from a moist bare soil surface (mm/day)
- Penman potential transpiration from a crop canopy (mm/day)
- daily global radiation in KJ/m2/day
- snow depth (cm) (data with no quality check)

A.1.3 SVP

A.1.3.1 Data Provenance


A.1.3.2 License

The Data may be used in the context of the ADMIRE project, for scientific purposes. Any distribution to parties outside of the project, or any commercial use is prohibited, and would require explicit approval by the Provider.

Also, the Provider is to be credited as the original source of the Data, and also is entitled to receive copies of all deliverables of the project ADMIRE, in which the Data or derived products play significant role – i.e. those deliverables, whose contents or meaning would change significantly, should the Data or derived products be withdrawn from them.

A.1.3.3 Data Structure

The data contains measurements of certain properties of water in several installations (hydroelectric dams), mainly the water level in the reservoir, temperature (also the air temperature), and the flow volumes. The Data is available for all months of years 1998 to 2007, and for first 5 months of year 2008. It covers 16 separate waterworks.

The Data consists of records of only one type. Apart from containing the date and time of the record, its structure is as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Units</th>
<th>Periodicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservoir parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum allowed water</td>
<td>Meters above sea level</td>
<td>Hourly</td>
</tr>
<tr>
<td>Level</td>
<td>Meters above sea level</td>
<td>Hourly</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Maximum allowed water level</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current water level</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total water volume</td>
<td>Millions of m³</td>
<td>Not present in any record</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weather</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Water temperature °C</td>
<td>Daily</td>
<td></td>
</tr>
<tr>
<td>Air temperature °C</td>
<td>Daily/hourly</td>
<td></td>
</tr>
<tr>
<td>Rainfall</td>
<td>Millimeters accumulated in last 60 minutes</td>
<td>Hourly</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inflow</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>m³.s⁻¹</td>
<td>Not present in any record</td>
</tr>
<tr>
<td>River</td>
<td>m³.s⁻¹</td>
<td>Daily</td>
</tr>
<tr>
<td>Turbines</td>
<td>m³.s⁻¹</td>
<td>Not present in any record</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outflow</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>m³.s</td>
<td>Hourly</td>
</tr>
<tr>
<td>Turbines</td>
<td>m³.s</td>
<td>Hourly</td>
</tr>
<tr>
<td>Unused</td>
<td>m³.s</td>
<td>Not present in any record</td>
</tr>
<tr>
<td>Spillway</td>
<td>m³.s</td>
<td>Not present in any record</td>
</tr>
<tr>
<td>Aux. Spillway</td>
<td>m³.s</td>
<td>Not present in any record</td>
</tr>
<tr>
<td>Lock</td>
<td>m³.s</td>
<td>Not present in any record</td>
</tr>
<tr>
<td>Biological</td>
<td>m³.s</td>
<td>Hourly</td>
</tr>
<tr>
<td>Draw-off</td>
<td>m³.s</td>
<td>Not present in any record</td>
</tr>
</tbody>
</table>

Table 14: Structure of the SVP data set

A.1.4 SNOW

A.1.4.1 Data Provenance

The data has been provided by the Slovak Water Enterprise and it is the product of SWE’s own daily measurements.

A.1.4.2 License

The data has been provided to UISAV for research purposes and may be used in context of the project ADMIRE by UISAV and its project partners.

A.1.4.3 Data Structure

The data contains two sets of pairs of values (one set for each reservoir). Each pair consists of a date and the total amount of water in the snow cover in the watershed area of the respective reservoir. The period between two consecutive measurements is 14 days. The amount of water is given in m³.
A.1.4.4 INFLOW

This batch of data contains the historical inflows of the two selected water reservoirs in the Vah river watershed area: Liptovská Mara and Oravská priehrada. The data is being used in the SVP scenario.

A.1.4.5 Data Provenance

The data has been provided by the Slovak Water Enterprise, even though it originated in the measurement network of the Slovak Hydro-meteorological Institute.

A.1.4.6 License

The data has been provided to UISAV for research purposes and may be used in context of the project ADMIRE by UISAV and its project partners.

A.1.4.7 Data Structure

The data contains daily values of mean inflow into the reservoir in the structure:

- Reservoir name
- Date of measurement
- Value of mean inflow in m$^3$.s$^{-1}$

A.1.5 RADAR

This batch contains both raw data from weather radars and their graphical representation. The data represents potential rainfall amounts in mm of water per hour.

A.1.5.1 Data Provenance

The data has been provided by the Slovak Hydro-meteorological Institute.

A.1.5.2 License

The data and models may be used for research purposes by the partners of the ADMIRE project. Use for commercial purposes, as well as use outside of the scope of the ADMIRE project is not allowed.

A.1.5.3 Data Structure

The data contains two separate batches:

- Raw radar data
  
  The data consists of text files with values for grid points. The files are in a period of 15 minutes, meaning there are 4 files in a hour, and 96 files in one day. The geographical coordinates for the grid points are described in a separate text file (one for the whole batch).

  The data represents actual atmospheric reflectivity measured by the weather radar.

- Weather radar imagery
  
  The data consists of text files with values for grid points. The files are in a period of 15 minutes, meaning there are 4 files in a hour, and 96 files in one day. The geographical coordinates for the grid points are described in a separate text file (one for the whole batch).

  The data represents potential rainfall, in mm of water accumulated per hour.
A.1.6 SHMU_HYDRO

This dataset contains historical values of water temperature, water flow volume, and water height measured by several hydrological measurement stations deployed in Orava (part of Slovakia). Figure 6 shows the deployment of the stations.

A.1.6.1 Data Provenance

The data has been provided by the Slovak Hydro-meteorological Institute from their hydrological department’s database.

A.1.6.2 License

The data and models may be used for research purposes by the partners of the ADMIRE project. Use for commercial purposes, as well as use outside of the scope of the ADMIRE project is not allowed.

A.1.6.3 Data Structure

The data contains three different parameters for a number of hydrological stations. The parameters are:

- Water temperature – the immediate temperature of the water at the station;
- Water height – the immediate water height at the station;
- Water flow volume – the volume of the water flowing through the river at the station’s point, averaged for the last hour.

The dataset also contains metadata, describing the stations themselves:

- Geographical location of the station;
- Stream which the station measures;
- Name of the station;
- ID (numerical value shown in Figure 41) of the station;
- Starting time of the measured data from the station;
- End time of the measured data from the station.
A.2 Custom Processing Elements of Flood Forecasting Simulation Cascade Scenarios

**eu.admire.spatioTemporal.activities.grib.GribCellValues**

**Input:**
- gribFiles: list of file names containing the required data
- gribCells: list of IDs of cells which contain the required data for the selected location

**Output:**
- gribResult: list of tuples containing the data

**Function:** reading data from selected cells from GRIB files

**eu.admire.spatioTemporal.activities.tupleMerging.OrderedTuplesMerge**

**Input:**
- tuple1, tuple2: input tuples which have to merge
- tuplepos, tuple2pos: positions of the columns to be sorted

**Output:**
- result: merged tuples with order according to columns tuple1pos, tuple2pos

**Function:** merging data with respect to the order specify in tuple1pos, tuple2pos (mostly according to time). This PE is important because of occasional missing data from input, which can cause wrong order when merged with standard SimpleTupleMerge in OGSA-DAI.

**eu.admire.LinearTrendFilter**
Input:

data: input data (in form of list of tuples)
linearIndex: the index of column which is needed to interpolate for missing values

Output:

result: the data with interpolated values for missing data.

Function: some data are measured with other frequency than other, so the missing data are needed to interpolate.

**eu.admire.BuildClassifierLinearRegression**

Input:

data: the data for training
classIndex: the column which will be predicted
columnIndices, nominalValues:

Output:

classifier: the trained model

Function: this PE will create a classifier according to the input data.

**eu.admire.spatioTemporal.activities.radar.GetFileNames**

Input:

start, end: time of beginning and end of time period we need to integrate data
step: frequency of data in minutes (typically 15 or 60min)

Output:

Output: list of files containing the radar data for give period and frequency

Function: selecting radar data files according to time period and frequency

**eu.admire.spatioTemporal.activities.radar.ReadData**

Input:

files: list of files containing data
lat, lon: geographical coordination of the selected location (latitude, longitude)

Output:

output: radar data with size 10x10 around the selected location

Function: reading data from the selected location from the radar data files given in input.

Beside the mentioned PEs, the scenarios also use several standard PEs provided by OGSA-DAI or ADMIRE framework:
uk.org.ogsadai.SQLQuery
uk.org.ogsadai.TupleToWebRowSetCharArrays
uk.org.ogsadai.WebRowSetCharacterDataToTuple
eu.admire.Results
uk.org.ogsadai.TupleSimpleMerge
uk.org.ogsadai.TupleArithmeticProject
eu.admire.Serialiser
eu.admire.Deserialiser
uk.org.ogsadai.DeliverToFTP
uk.org.ogsadai.ObtainFromFTP
Appendix B  DISPEL Documents for Application Scenarios

B.1 Cross-selling Scenario

```java
use uk.org.ogsadai.SQLQuery;
use eu.admire.ItemList;
use eu.admire.Results;
use eu.admire.BuildIterativeAssociator;
use eu.admire.BuildAssociator;
use uk.org.ogsadai.ControlledRepeat;
use uk.org.ogsadai.DeliverToFTP;
use uk.org.ogsadai.DeliverToNull;
use eu.admire.AssociatorPMML;
use eu.admire.PMMLSVG;
use uk.org.ogsadai.Tee;
use eu.admire.AssociatorXml;
use uk.org.ogsadai.TupleToWebRowSetCharArrays;
use uk.org.ogsadai.TupleSelect;

// This function delivers single result to the Repository
PE(<Connection data> => <>()) repeatedDelivery(String host, String fileName, Boolean passive) {
    ControlledRepeat repeat1 = new ControlledRepeat;
    ControlledRepeat repeat2 = new ControlledRepeat;
    ControlledRepeat repeat3 = new ControlledRepeat;
    DeliverToFTP dftp = new DeliverToFTP;
    DeliverToNull dnull = new DeliverToNull;
    repeat2.output => dftp.filename;  // This function produce sample of data to be shown in chart visualizer
    PE(<Connection data> => <>()) sampleResult(String condition) {
        Results result1 = new Results;
        TupleSelect select = new TupleSelect;
        TupleToWebRowSetCharArrays ttwrs = new TupleToWebRowSetCharArrays;
        select.result => ttwrs.data;  // This function produce three results from one associator and submits all of them to
        return PE(<Connection data=select.data> => <>());
    }
    return PE(<Connection data=repeat1.input> => <>());
}
```

// This function produce three results from one associator and submits all of them to
// Repository using deliverRules() function:
// xml representation
// pmml representation
// svg visualisation
ADMIRE-REPORT on Pilot Applications Deployment and Platform Evaluation D6.6

PUBLIC

PE(<Connection data> => <>) deliverRules(String baseFileName, String host, Boolean passive) {
    PE(<Connection data> => <>) RepeatedDelivery1 = repeatedDelivery(host, baseFileName + ".ar", true);
    RepeatedDelivery1 dr1 = new RepeatedDelivery1;
    PE(<Connection data> => <>) RepeatedDelivery2 = repeatedDelivery(host, baseFileName + ".ar.pml", true);
    RepeatedDelivery2 dr2 = new RepeatedDelivery2;
    PE(<Connection data> => <>) RepeatedDelivery3 = repeatedDelivery(host, baseFileName + ".ar.svg", true);
    RepeatedDelivery3 dr3 = new RepeatedDelivery3;
    AssociatorPMML apmml = new AssociatorPMML;
    AssociatorXml axml = new AssociatorXml;
    PMMLSVG pmmlsvg = new PMMLSVG;
    Tee t = new Tee;
    t.output[0] => apmml.associator;
    t.output[1] => axml.associator;
    axml.xml => dr1.data;
    apmml.pml => dr2.data;
    apmml.pmml => pmmlsvg.pmml;
    pmmlsvg.svg => dr3.data;
    return PE(<Connection data=t.input> => <>);
}

SQLQuery getTransactions = new SQLQuery;
SQLQuery getProducts = new SQLQuery;
ItemList itemList = new ItemList;
BuildAssociator associator = new BuildAssociator;
TupleToWebRowSetCharArrays ttwrss2 = new TupleToWebRowSetCharArrays;
Results result1 = new Results;
Results result2 = new Results;
Results result3 = new Results;

String expression1 = "SELECT ID, NAME FROM PC_PRODUCTS_T WHERE ID BETWEEN 0 and 26";
String expression2 = "SELECT c.CUSTOMER_ID, p.NAME FROM CDM_CONTRACT_ITEMS_T c JOIN PC_PRODUCTS_T p ON c.PRODUCT_ID = p.ID WHERE c.CUSTOMER_ID < 1000";

[- expression2 -] => getTransactions.expression;
[- "crm" -] => getTransactions.resource;
[- expression1 -] => getProducts.expression;
[- "crm" -] => getProducts.resource;
getTransactions.data => itemList.transactions;
getProducts.data => itemList.items;
[- 0 -] => itemList.keyIndex;
itemList.list => associator.data;

for (Integer i=0;i<26;i++) {
    [- "$f", "$t" -] => associator.nominalValues[i];
}
[- "$I 0 -N 10 -T 0 -C 0.0 -D 0.00 -S 0.0 -U 1.0 -M 0.01* -| => associator.options;
[- "weka.associations.FPGrowth" -] => associator.algorithmClass;

// Uncomment lines below to deliver results to repository.
//PE(<Connection data> => <>) DeliverRules = deliverRules("rules", "maciekjarka:cocacola@localhost", true);
//DeliverRules dr = new DeliverRules;
//associator.associator => dr.data;

// Uncomment lines below to add chart visualiser result available
// SampleResult1 sr = new SampleResult1;  
//getTransactions.data => sr.data;

// This result can be displayed in text viewer
associator.associator => result2.input;
|- "associator" -| => result2.name;

associator.rules => ttwrs2.data;
// This result can be displayed in table viewer
ttwrs2.result => result1.input;
|- "rules" -| => result1.name;

// itemList.list => result3.input;

//PE(<Connection data> => <>) SampleResult1 = sampleResult("CUSTOMER_ID > 0 AND CUSTOMER_ID < 1000");
//SampleResult1 sr = new SampleResult1;
//getTransactions.data => sr.data;

B.2  ORAVA scenario – Data integration

use uk.org.ogsadai.SQLQuery;
use uk.org.ogsadai.TupleToWebRowSetCharArrays;
use eu.admire.spatioTemporal.activities.tupleMerging.OrderedTuplesMerge;
use eu.admire.spatioTemporal.activities.grib.GribCellValues;
use uk.org.ogsadai.TupleSimpleMerge;
use eu.admire.DataTransfer;
use eu.admire.Results;
use eu.admire.LinearTrendFilter;
use uk.org.ogsadai.TupleArithmeticProject;
use eu.admire.BuildClassifierLinearRegression;
use eu.admire.Classify;
use eu.admire.Serialiser;
use uk.org.ogsadai.DeliverToFTP;
use uk.org.ogsadai.TupleToCSV;

// Composite PE for reading data from Grib.
// Parameters:
// startDate, endDate: start and end date of duration, in form yyyy-mm-dd
// value: code of data from grib (CWDI for precipitation and TMP for temperature)
// longitude, latitude: longitude and latitude of selected location (Orava waterwork or water stations along river)
PE(< => <Connection output>) readingGribFunction(String startDate, String endDate, String value, String longitude, String latitude)
{
    GribCellValues grib = new GribCellValues;
    SQLQuery gribFileSelector = new SQLQuery;
    SQLQuery gribCoordsSelector = new SQLQuery;

    String gribID = "DbGribMetaResource";

    String gribFileSelectQuery = "SELECT file, date_time, type, type_desc,
     type_unit FROM grib_meta_c WHERE type='" + value + "' and date_time >=" + startDate + " 00:00:00' and date_time <=" + endDate + " 24:00:00' and forecast=0 and grid_x_size=281 order by date_time;"
    String gridCoordSelectQuery = "SELECT id AS ID, lat, lon, ( sqrt( (lon-" + longitude + ")^2) + (lat-" + latitude + ")^2) ) AS dist FROM grid_coords2 order by dist limit 0,1;";
ADMIRE-REPORT ON PILOT APPLICATIONS DEPLOYMENT AND PLATFORM EVALUATION D6.6

PUBLIC

-- gribFileSelectQuery -| => gribFileSelector.expression;
|-- gribID -| => gribFileSelector.resource;

|-- gridCoordSelectQuery -| => gribCoordsSelector.expression;
|-- gribID -| => gribCoordsSelector.resource;

gribFileSelector.data => grib.gribFiles;
gribCoordsSelector.data => grib.gribCells;

return PE (<> => <Connection output = grib.gribResult> );

// Parameters of DISPEL, should be changed for every execution
String start_date = "2006-01-01";
String end_date = "2006-01-31";
String station_id = "5848";
String station_lon = "19.55";
String station_lat = "49.3355";

// Location of Orava waterwork, should not modify
String orava_lon = "19.548";
String orava_lat = "49.3366";
String repository_file = "data_CPE_" + start_date + "_" + end_date + "_" + station_id + ".xml";
String model_file = "classifier_" + station_id;

// Reading precipitation from Orava
PE(<> => <Connection output>) readingGribOrava = readingGribFunction(start_date, end_date,
"CWDI", orava_lon, orava_lat);
readingGribOrava oravaPE = new readingGribOrava;

// Setting very small values 1.0e-8 to zero
TupleArithmeticProject gribArithProject = new TupleArithmeticProject();
oravaPE.output => gribArithProject.data;
|-- [ "ZeroEpsilon(val, 0.000000000001)", "date_time" ] -| => gribArithProject.expressions;
|-- [ "rainFall", "date_time1" ] -| => gribArithProject.resultColumnNames;

// Reading temperature from Orava
PE(<> => <Connection output>) readingGribOravaTmp = readingGribFunction(start_date, end_date,
"TMP", orava_lon, orava_lat);
readingGribOravaTmp oravaTmpPE = new readingGribOravaTmp;

// Change Kelvin temperature to Celsius
TupleArithmeticProject gribArithProject2 = new TupleArithmeticProject();
oravaTmpPE.output => gribArithProject2.data;
|-- [ "val-273.15", "date_time" ] -| => gribArithProject2.expressions;
|-- [ "airTemp", "date_time2" ] -| => gribArithProject2.resultColumnNames;

// Reading precipitation from water station
PE(<> => <Connection output>) readingGribStation = readingGribFunction(start_date, end_date,
"CWDI", station_lon, station_lat);
readingGribStation stationPE = new readingGribStation;

// Setting very small values 1.0e-8 to zero
TupleArithmeticProject gribArithProject3 = new TupleArithmeticProject();
stationPE.output => gribArithProject3.data;
|-- [ "ZeroEpsilon(val, 0.000000000001)", "date_time" ] -| => gribArithProject3.expressions;
|-- [ "rainFallStation", "date_time3" ] -| => gribArithProject3.resultColumnNames;
// Extract reservoir data

SQLQuery oravaReservoirData = new SQLQuery();

// form connection cl with an explicit literal stream expression as its source
// and query as its destination teplota_vzduch, odtok_cel_sucet

String waterWorkQuery = "SELECT vodne_dielo, datum, teplota_vzduch, teplota_voda, odtok_cel_sucet FROM `new_vodne_diela`
WHERE vodne_dielo='Tvrdošín' and datum>='" + start_date + " 00:00:00' and datum <='" + end_date + " 24:00:00'
order by datum;";

String waterworkDb = "DbSvpResource";

|- waterWorkQuery => oravaReservoirData.expression;
|- waterworkDb => oravaReservoirData.resource;

// Linear filter teplota_voda column
LinearTrendFilter oravaReservoirLinearTrendFilter = new LinearTrendFilter();

oravaReservoirData.data => oravaReservoirLinearTrendFilter.data;

3 => oravaReservoirLinearTrendFilter.linearIndex;

// merge results from above workflows

OrderedTuplesMerge mergel = new OrderedTuplesMerge();

gribArithProject.result => mergel.tuple1;

oravaReservoirLinearTrendFilter.result => mergel.tuple2;

Integer number1 = 1;

Integer number2 = 1;

number1 => mergel.tuple1pos;

number2 => mergel.tuple2pos;

// Station discharge

SQLQuery stationDischargeData = new SQLQuery();

String stationDischargeQuery = "SELECT id, timestamp(day, time) as date, value as Dicharge FROM `Qh` WHERE `id` = " + station_id + " and day>='" + start_date + "' and day<='" + end_date + "' and concat(day,time)>='" + start_date + "00:00:00' and concat(day,time)<='" + end_date + "24:00:00' order by day asc, time asc;";

String waterStationDB = "DbOravaWaterStationsResource";

stationDischargeQuery => stationDischargeData.expression;

waterStationDB => stationDischargeData.resource;

// Station Water Temperature

SQLQuery stationWaterTempData = new SQLQuery();

String stationWaterTempQuery = "SELECT id, timestamp(day, time) as date, value as Temp FROM `Thod` WHERE `id` = " + station_id + " and day>='" + start_date + "' and day<='" + end_date + "' and concat(day,time)>='" + start_date + "00:00:00' and concat(day,time)<='" + end_date + "24:00:00' order by day asc, time asc;";

stationWaterTempQuery => stationWaterTempData.expression;

waterStationDB => stationWaterTempData.resource;

// Station Water Level

SQLQuery stationWaterLevelData = new SQLQuery();

String stationWaterLevelQuery = "SELECT id, timestamp(day, time) as date, value as Level FROM `Hh` WHERE `id` = " + station_id + " and day>='" + start_date + "' and day<='" + end_date + "' and concat(day,time)>='" + start_date + "00:00:00' and concat(day,time)<='" + end_date + "24:00:00' order by day asc, time asc;";
// merge results from water station database

OrderedTuplesMerge merge2 = new OrderedTuplesMerge();
stationDischargeData.data => merge2.tuple1;
stationWaterTempData.data => merge2.tuple2;
| number2 | => merge2.tuple1pos;
| number2 | => merge2.tuple2pos;

OrderedTuplesMerge merge3 = new OrderedTuplesMerge();
merge2.result => merge3.tuple1;
stationWaterLevelData.data => merge3.tuple2;
| number2 | => merge3.tuple1pos;
| number2 | => merge3.tuple2pos;

OrderedTuplesMerge merge4 = new OrderedTuplesMerge();
merge1.result => merge4.tuple1;
merge3.result => merge4.tuple2;
| number1 | => merge4.tuple1pos;
| number2 | => merge4.tuple2pos;

OrderedTuplesMerge merge5 = new OrderedTuplesMerge();
gribArithProject2.result => merge5.tuple2;
| number1 | => merge5.tuple1pos;
| number1 | => merge5.tuple2pos;

OrderedTuplesMerge merge6 = new OrderedTuplesMerge();
gribArithProject3.result => merge6.tuple2;
| number1 | => merge6.tuple1pos;
| number1 | => merge6.tuple2pos;

// Final project
TupleArithmeticProject finalProject = new TupleArithmeticProject();
merge6.result => finalProject.data;

TupleToWebRowSetCharArrays csv = new TupleToWebRowSetCharArrays;
finalProject.result => csv.data;

//This should really be using the repository to save the classifier.
DeliverToFTP ftp = new DeliverToFTP;
csv.result => ftp.data;
| repository_file | => ftp.filename;
| "admire_repository:admirerepos887@hicks.ui.sav.sk" | => ftp.host;

submit ftp;
B.3 ORAVA scenario – Training

1. /* use non-universal components from the computational environment */
2. use uk.org.ogsadai.SQLQuery; //get definition of SQLQuery
3. use uk.org.ogsadai.TupleToWebRowSetCharArrays; // serialisation
4. use eu.admire.DataTransfer;
5. use eu.admire.Results;
6. use eu.admire.BuildClassifierLinearRegression;
7. use eu.admire.BuildClassifierM5P;
8. use eu.admire.Classify;
9. use eu.admire.Serialiser;
10. use uk.org.ogsadai.DeliverToFTP;
11. use uk.org.ogsadai.ObtainFromFTP;
12. use uk.org.ogsadai.WebRowSetCharacterDataToTuple;
13. use uk.org.ogsadai.CSVToTuple;
14. ObtainFromFTP ftp = new ObtainFromFTP; //filename, host, data
15. |- "data-wrs" -| => ftp.filename;
16. |- "admire_repository:admirerepos887@hicks.ui.sav.sk" -| => ftp.host;
17. WebRowSetCharacterDataToTuple reader = new WebRowSetCharacterDataToTuple;
18. ftp.data => reader.data;
19. BuildClassifierLinearRegression buildClassifier = new BuildClassifierLinearRegression();
20. reader.result => buildClassifier.data;
21. |- [ ] -| => buildClassifier.columnIndices;
22. |- [ ] -| => buildClassifier.nominalValues;
23. |- 7 -| => buildClassifier.classIndex; // the value we are trying to predict
24. //|- "-C 0.25 -M 2" -| => buildClassifier.options;
25. //|- 10 -| => buildClassifier.folds;
26. //Results stat = new Results;
27. //buildClassifier.evaluation => stat.input;
28. //|- "statistics" -| => stat.name;
29. Serialiser ser = new Serialiser;
30. buildClassifier.classifier => ser.data;
31. //This should really be using the repository to save the classifier.
32. DeliverToFTP ftp2 = new DeliverToFTP;
33. ser.result => ftp2.data;
34. |- "classifier" -| => ftp2.filename;
35. |- "admire_repository:admirerepos887@hicks.ui.sav.sk" -| => ftp2.host;
36. submit ftp2;

B.4 ORAVA scenario – Prediction

1. /* use non-universal components from the computational environment */
2. use uk.org.ogsadai.SQLQuery; //get definition of SQLQuery
3. use uk.org.ogsadai.TupleToWebRowSetCharArrays; // serialisation
4. use eu.admire.spatioTemporal.activities.tupleMerging.OrderedTuplesMerge;
5. use eu.admire.spatioTemporal.activities.grib.GribCellValues; //Grib processing element
6. use uk.org.ogsadai.TupleSimpleMerge;
7. use eu.admire.DataTransfer;
8. use eu.admire.Results;
9. use eu.admire.LinearTrendFilter;
10. use uk.org.ogsadai.TupleArithmeticProject;
11. use eu.admire.BuildClassifierLinearRegression;
12. use eu.admire.BuildClassifierM5P;
13. use eu.admire.Classify;
14. //use uk.org.ogsadai.Head;
15. use uk.org.ogsadai.DeliverToNull;
16. use eu.admire.Serialiser;
17. use uk.org.ogsadai.DeliverToFTP;
18. use eu.admire.Deserialiser;
19. use uk.org.ogsadai.WebRowSetCharacterDataToTuple;
20. use uk.org.ogsadai.ObtainFromFTP;

21. ObtainFromFTP ftp = new ObtainFromFTP; //filename, host, data
22. |-> "data-wrs" | => ftp.filename;
23. |-> "admire_repository:admirerepos887@hicks.ui.sav.sk" | => ftp.host;

24. WebRowSetCharacterDataToTuple reader = new WebRowSetCharacterDataToTuple;
25. ftp.data => reader.data;

26. ObtainFromFTP ftp2 = new ObtainFromFTP; //filename, host, data
27. |-> "clasifier" | => ftp2.filename;
28. |-> "admire_repository:admirerepos887@hicks.ui.sav.sk" | => ftp2.host;

29. Deserialiser des = new Deserialiser;
30. ftp2.data => des.data;

31. Classify class1 = new Classify;
32. des.result => class1.classifier;
33. reader.result => class1.data;

34. // project to get the actual value
35. TupleArithmeticProject actualValueProject = new TupleArithmeticProject;
36. reader.result => actualValueProject.data;
37. //|-> [ "Level" ] | => actualValueProject.expressions;
38. |-> [ "waterTempAtStation" ] | => actualValueProject.expressions;
39. |-> [ "actual" ] | => actualValueProject.resultColumnNames;

40. // Now see the actual and expected in one tuple - THESE SEEM TO BE BEING MERGED IN THE
41. // WRONG ORDER
42. TupleSimpleMerge actualAndPredMerge = new TupleSimpleMerge;
43. actualValueProject.result => actualAndPredMerge.data[0];
44. class1.result => actualAndPredMerge.data[1];

45. TupleToWebRowSetCharArrays wrs = new TupleToWebRowSetCharArrays;
46. actualAndPredMerge.result => wrs.data;
47. Results result = new Results;
48. |-> "actual and predicted" | => result.name;
49. wrs.result => result.input;
50. submit wrs;
B.5 RADAR Scenario – Motion vector based radar image computation (prediction) with subsequent precipitation forecast

```java
package eu.admire.demo.radar {

use uk.org.ogsadai.SQLQuery;
use uk.org.ogsadai.TupleToWebRowSetCharArrays;
use eu.admire.Results;
use eu.admire.spatioTemporal.activities.radar.SelectRadarFiles;
use eu.admire.spatioTemporal.activities.radar.ReadRawRadarData;
use eu.admire.spatioTemporal.activities.radar.CalculateNextImages;
use eu.admire.spatioTemporal.activities.radar.RadarDataSpaceSync;
use uk.org.ogsadai.TupleToCSV;
use uk.org.ogsadai.TupleSplit;
use uk.org.ogsadai.DeliverToNull;
use uk.org.ogsadai.TupleArithmeticProject;
use eu.admire.BuildClassifier;
use eu.admire.Serialiser;
use uk.org.ogsadai.ObtainFromFTP;
use uk.org.ogsadai.TupleSimpleMerge;
use eu.admire.Classify;
use eu.admire.Deserialiser;
use uk.org.ogsadai.GenericTupleTransform;
use uk.org.ogsadai.WebRowSetCharacterDataToTuple;
use uk.org.ogsadai.DeliverToFTP;
use eu.admire.spatioTemporal.activities.radar.RadarImageVis;

String start_date = "2007-01-18 12:00:00";
String end_date = "2007-01-18 13:00:00";
String model_name = "radar_classifier";

// Reading radar data

SelectRadarFiles filenames = new SelectRadarFiles;
| - start_date - | => filenames.start;
| - end_date - | => filenames.end;
| - "15" - | => filenames.step;

ReadRawRadarData rawradar = new ReadRawRadarData;
filenames.output => rawradar.files;

CalculateNextImages nextradar = new CalculateNextImages;
rawradar.output => nextradar.data;
| - "3" - | => nextradar.forecast;

ObtainFromFTP ftp = new ObtainFromFTP; //filename, host, data
| - "stations.xml" - | => ftp.filename;
| - "admire_repository:admirerepos887@hicks.ui.sav.sk" - | => ftp.host;

WebRowSetCharacterDataToTuple reader = new WebRowSetCharacterDataToTuple;
ftp.data => reader.data;

RadarDataSpaceSync merge = new RadarDataSpaceSync;
nextradar.output => merge.radar;
reader.result => merge.precipitation;
| - "no" - | => merge.timeCheck;

```
new TupleSimpleMerge actualAndPredMerge = new TupleSimpleMerge; 
merge.output => actualAndPredMerge.data[0];
class1.result => actualAndPredMerge.data[1];

TupleToCSV toWRS = new TupleToCSV; 
actualAndPredMerge.result => toWRS.data;
true => toWRS.includeHeader;

DeliverToFTP ftp4 = new DeliverToFTP; 
toWRS.result => ftp4.data;
"radar-prediction" => ftp4.filename;
"admire_repository:admirerepos887@hicks.ui.sav.sk" => ftp4.host;

TupleSplit splitter = new TupleSplit; 
nextradar.output => splitter.data;

DeliverToNull discard = new DeliverToNull; 
splitter.result[0] => discard.input;

RadarImageVis vis = new RadarImageVis; 
splitter.result[1] => vis.data;

DeliverToFTP ftp5 = new DeliverToFTP; 
vis.output => ftp5.data;
"radar1.png", "radar2.png", "radar3.png" => ftp5.filename;
"admire_repository:admirerepos887@hicks.ui.sav.sk", 
"admire_repository:admirerepos887@hicks.ui.sav.sk", 
"admire_repository:admirerepos887@hicks.ui.sav.sk" => ftp5.host;

submit ftp5;
submit ftp4;
}

B.6  SVP Scenario – Data integration

// This is DISPEL script for data integration of SVP scenario
// Viet Tran, IISAS

use uk.org.ogsadai.SQLQuery;
use uk.org.ogsadai.TupleToWebRowSetCharArrays;
use eu.admire.Results;
use uk.org.ogsadai.TupleToCSV;
use uk.org.ogsadai.TupleArithmeticProject;
use eu.admire.BuildClassifier;
use eu.admire.Serialize;
use uk.org.ogsadai.DeliverToFTP;
use uk.org.ogsadai.GenericTupleTransform;
use eu.admire.spatioTemporal.activities.svp.DailyAggregation;
use eu.admire.spatioTemporal.activities.tupleMerging.OrderedTuplesMerge;

// Parameters of SVP scenario
String start_date = "2000-01-03";
String end_date = "2007-12-30";
String repository_file = "svp-data-integration-" + start_date + "+" + end_date + ".csv";

// Query data from waterworks
SQLQuery query = new SQLQuery;
String expression = "SELECT datum, teplota_vzduch, zrazky from new_vodne_diela where vodne_dielo='Orava' and datum>= " + start_date + " 08:00:00' and datum <= " + end_date + " 07:00:00' order by datum";

// Aggregation of hourly data to daily (temperatures and precipitation)
DailyAggregation agg = new DailyAggregation;
query.data => agg.data;

// Query data from snow database
SQLQuery query2 = new SQLQuery;
String expression2 = "SELECT date, snow from svp_snow where station='Orava' and date>= " + start_date + " and date <= " + end_date + " order by date";

// Merging data with respect to date
OrderedTuplesMerge merge4 = new OrderedTuplesMerge();
query2.data => merge4.tuple2;
agg.output => merge4.tuple1;
- 0 -| => merge4.tuple1pos;
- 0 -| => merge4.tuple2pos;

// **********************************************
// Merging available data
// **********************************************
OrderedTuplesMerge merge5 = new OrderedTuplesMerge();
query3.data => merge5.tuple2;
merge4.result => merge5.tuple1;
|- 0 -| => merge5.tuplepos;
|- 0 -| => merge5.tuple2pos;

// *****************************************************

// Data projection: select significant columns (and possibly change column name)
TupleArithmeticProject finalProject = new TupleArithmeticProject();
merge5.result => finalProject.data;
|- [ "datum", "teplota_vzduch", "zrazky", "snow", "inflow"] -| => finalProject.expressions;
|- [ "datum", "teplota_vzduch", "zrazky", "snow", "inflow"] -| =>
finalProject.resultColumnNames;

// Filter: Select only interesting rows and discard others
// This is a ruby script for filter as a string
String filter_script = "
def getMetadata(inputMetadata)
  inputMetadata;
end
$max_null = 60
$current_null = 61
$list = nil
def process(tuple)
  value = tuple.getObject(3)
  if (value != Java::uk.org.ogsadai.tuple.Null.getValue)
    if ($list == nil)
      $list = Java::java.util.ArrayList.new
    end
    $list.add(tuple)
    $current_null = 0;
    new_list = $list;
    $list = nil;
    new_list;
  elsif ($current_null < $max_null)
    if ($list == nil)
      $list = Java::java.util.ArrayList.new
    end
    $list.add(tuple)
    $current_null = $current_null + 1;
    Java::java.util.Collections.emptyList;
  else
    $list = nil;
    Java::java.util.Collections.emptyList;
  end
end

def flush()
  Java::java.util.Collections.emptyList;
end
";
// End of ruby script
// Perform filtering
GenericTupleTransform filter = new GenericTupleTransform;
B.7 SVP Scenario – Training

use uk.org.ogsadai.TupleToWebRowSetCharArrays; // serialisation
use eu.admire.Serialiser;
use uk.org.ogsadai.DeliverToFTP;
use uk.org.ogsadai.ObtainFromFTP;
use uk.org.ogsadai.CSVToTuple;
use eu.admire.spatioTemporal.activities.svp.PreProcessing;
use eu.admire.spatioTemporal_activities.svp.TrainingFlow;
use eu.admire.spatioTemporal_activities.svp.TrainingSnow;

// Parameters of SVP scenario
String start_date = "2000-01-03";
String end_date = "2007-12-30";
String repository_file = "svp-data-integration-" + start_date + "_" + end_date + ".csv";

ObtainFromFTP ftp = new ObtainFromFTP; //filename, host, data
f| | "repository_file" => ftp.filename;
| | "admire_repository:admirerepos887@hicks.ui.sav.sk" => ftp.host;

CSVToTuple reader = new CSVToTuple;
| true => reader.headerIncluded;
ftp.data => reader.data;

PreProcessing proproces = new PreProcessing;
reader.result => proproces.data;

TrainingFlow flowModel = new TrainingFlow;
proproces.output => flowModel.data;

Serialiser serl = new Serialiser;
flowModel.output => serl.data;

//This should really be using the repository to save the classifier.
DeliverToFTP ftp1 = new DeliverToFTP;
serl.result => ftp1.data;
| "flowModel" => ftp1.filename;
| "admire_repository:admirerepos887@hicks.ui.sav.sk" => ftp1.host;

TrainingFlow snowModel = new TrainingSnow;
preproces.output => snowModel.data;

Serialiser ser2 = new Serialiser;
snowModel.output => ser2.data;

//This should really be using the repository to save the classifier.
DeliverToFTP ftp2 = new DeliverToFTP;
ser2.result => ftp2.data;
|- "snowModel" -| => ftp2.filename;
|- "admire_repository:admirerepos887@hicks.ui.sav.sk" -| => ftp2.host;

submit ftp1;
submit ftp2;

**B.8 SVP Scenario – Prediction**

use uk.org.ogsadai.TupleToWebRowSetCharArrays; // serialisation
use eu.admire.Serialiser;
use eu.admire.Deserialiser;
use uk.org.ogsadai.DeliverToFTP;
use uk.org.ogsadai.ObtainFromFTP;
use uk.org.ogsadai.CSVToTuple;
use uk.org.ogsadai.TupleToCSV;
use eu.admire.spatioTemporal.activities.svp.PreProcessing;
use eu.admire.spatioTemporal.activities.svp.TrainingFlow;
use eu.admire.spatioTemporal.activities.svp.TrainingSnow;
use eu.admire.spatioTemporal.activities.svp.Prediction;
use uk.org.ogsadai.TupleArithmeticProject;
use uk.org.ogsadai.TupleSimpleMerge;

// Parameters of SVP scenario
String start_date = "2007-01-07";
String end_date = "2007-01-15";
String repository_file = "svp-data-integration-" + start_date +"_" + end_date + ".csv";

ObtainFromFTP ftp1 = new ObtainFromFTP; //filename, host, data
|- "flowModel" -| => ftp1.filename;
|- "admire_repository:admirerepos887@hicks.ui.sav.sk" -| => ftp1.host;

Deserialiser des1 = new Deserialiser;
ftp1.data => des1.data;

ObtainFromFTP ftp2 = new ObtainFromFTP; //filename, host, data
|- "snowModel" -| => ftp2.filename;
|- "admire_repository:admirerepos887@hicks.ui.sav.sk" -| => ftp2.host;

Deserialiser des2 = new Deserialiser;
ftp2.data => des2.data;

ObtainFromFTP ftp3 = new ObtainFromFTP; //filename, host, data
|- repository_file -| => ftp3.filename;
|- "admire_repository:admirerepos887@hicks.ui.sav.sk" -| => ftp3.host;

CSVToTuple reader = new CSVToTuple;
|- true -| => reader.headerIncluded;
ftp3.data => reader.data;
TupleArithmeticProject gribArithProject1 = new TupleArithmeticProject();
reader.result => gribArithProject1.data;
| [ "teplota_vzduch", "zrazky", "snow", "snow", "inflow", "inflow" ] -| =>
gribArithProject1.expressions;
| [ "temp", "rainfall", "snow", "snow_prev", "inflow_prev", "inflow" ] -| =>
gribArithProject1.resultColumnNames;

Prediction pred = new Prediction;
des1.result => pred.modelSnow;
des2.result => pred.modelFlow;
gribArithProject1.result => pred.data;

TupleSimpleMerge actualAndPredMerge = new TupleSimpleMerge;
reader.result => actualAndPredMerge.data[0];
pred.output => actualAndPredMerge.data[1];

TupleToCSV wrs = new TupleToCSV;
actualAndPredMerge.result => wrs.data;
| true -| => wrs.includeHeader;

DeliverToFTP ftp4 = new DeliverToFTP;
wrs.result => ftp4.data;
| "svp-prediction" -| => ftp4.filename;
| "admire_repository:admirerepos887@hicks.ui.sav.sk" -| => ftp4.host;

submit ftp4;

B.9 Seismology Scenario – Preprocessing

1. //The generic preprocessing stage
2. Type SeismoStage is PE(<Connection input;Connection stepbackid>=><Connection datasetid;Connection output;Connection metadata>)
3. Type SeismoPipeline is PE(<Connection input;Connection stepbackid>=><Connection [] metadata;Connection datasetid;Connection residue>)

4. //Builds the processing pipeline, new stages can be easily plugged in
5. PE <SeismoPipeline> makeArrayPipeline(PE <SeismoStage>[] TheStages) {
6.   Integer len = TheStages.length;
7.   SeismoStage[] stages = new SeismoStage[len];
8.   PE<SeismoStage> Stage = TheStages[0];
9.   stages[0] = new Stage;
10.  for (Integer i = 0; i<len-1; i++) {
11.     DeliverToNull tonull = new DeliverToNull;
12.     PE<SeismoStage> Stg = TheStages[i+1];
13.     stages[i+1] = new Stg;
14.     stages[i].output => stages[i+1].input;
15.     stages[i].datasetid => stages[i+1].stepbackid;
16.     stages[i].metadata=>tonull.input;
17.   };
18.   return SeismoPipeline( <Connection input = stages[0].input;
19.                             Connection stepbackid = stages[0].stepbackid> =>
20.                             <Connection metadata = stages[len-1].metadata;
21.                             Connection datasetid= stages[len-1].datasetid;
22.                             Connection residue = stages[len-1].output> );
23. };

DISPEL DOCUMENTS FOR APPLICATION SCENARIOS
24. // Extracts and synchronize time series based on the time window in input generates a 
25. WFExtractPE
26. PE<Connection window => <Connection output>) extractWFTimeSeries(String datares, 
27. String provenanceRes, String channel1Code, String station, String network)
28. {
29.   Tee tee = new Tee;
30.   ListConcatenate lc1 = new ListConcatenate;
31.   ControlledRepeat rpl_1 = new ControlledRepeat;
32.   WFRetrieve retrieve = new WFRetrieve;
33.   WaveformAppendAndSync appsyn = new WaveformAppendAndSync;
34.   SeismoMetadataTuple metastoreex1 = new SeismoMetadataTuple;
35.   SeismoMetadataTuple metastoreex2 = new SeismoMetadataTuple;
36.   metastoreex1.resource;
37.   metastoreex2.resource;
38.   Retrieve retrieve;
39.   WaveformAppendAndSync appsyn;
40.   SeismoMetadataTuple metastoreex1;
41.   SeismoMetadataTuple metastoreex2;
42.   DeliverToNull tonull1;
43.   DeliverToNull tonull2;
44.   datares;
45.   channel1Code;
46.   station;
47.   network;
48.   tee.output[0] => rpl_1.input;
49.   [-cha,sta,net] => rpl_1.repeatedInput;
50.   rpl_1.output => lc1.input[0];
51.   rpl_1.repeatedOutput => lc1.input[1];
52.   lc1.output => retrieve.parameters;
53.   retrieve.metadata => metastoreex1.metastring;
54.   metastoreex1.datasetid => metastoreex2.datasetid;
55.   return PE<Connection window=tee.input => <Connection output=appsyn.output;Connection 
56.   datasetid=metastoreex2.datasetid>);
57. }
58. //Instrument removal and normalization 
59. PE<SeismoStage> removeInstrumentMeanAndNormalize(String preprocGateway,String 
60. dataarea, String provenanceRes, String station, String channel, String metadataRes);
61. RespReader reader = new RespReader;
62. SeismoMetadataTuple metastoreex1 = new SeismoMetadataTuple;
63. SeismoMetadataTuple metastoreex2 = new SeismoMetadataTuple;
64. metastoreex1.resource;
65. metastoreex2.resource;
66. metadataRes;
67. cha=channel1Code;
68. sta=station;
69. net=network;
70. tee.output[0] => rpl_1.input;
71. [-cha,sta,net] => rpl_1.repeatedInput;
72. rpl_1.output => lc1.input[0];
73. rpl_1.repeatedOutput => lc1.input[1];
74. lc1.output => retrieve.parameters;
75. retrieve.metadata => metastoreex1.metastring;
76. metastoreex1.datasetid => metastoreex2.datasetid;
77. return PE<Connection window=tee.input => <Connection output=appsyn.output;Connection 
78. datasetid=metastoreex2.datasetid>);
79. }
66. ControlledRepeat rp = new ControlledRepeat;
67. reader.output=>rp.repeatedInput;

68. InstrumentCorrection inCorr = new InstrumentCorrection;
69. inCorr@gateway=preprocGateway;

70. rp.output=>inCorr.input;
71. WaveformNormalization normalize= new WaveformNormalization;
72. normalize@gateway=preprocGateway;

73. inCorr.output=>normalize.input;
74. DeliverToNull tonull = new DeliverToNull;
75. inCorr.mean=>normalize.parameters;
76. inCorr.metadata=>metastoreex1.metastring;
77. normalize.metadata=>metastoreex2.metastring;
78. metastoreex1.datasetid=>metastoreex2.stepbackid;

79. rp.repeatedOutput=> inCorr.responsepaz;
80. return PE(<Connection input=rp.input;Connection stepbackid=metastoreex1.stepbackid> =><Connection datasetid=metastoreex2.datasetid;Connection output=normalize.output;Connection metadata=metastoreex2.processedMetadata> );
81. };

82. //Signal filtering and decimation
83. PE<SeismoStage> filterAndDecimate(String preprocGateway,String datares,String provenanceRes){
84. SeismoMetadataTuple metastoreex1 = new SeismoMetadataTuple;
85. SeismoMetadataTuple metastoreex2 = new SeismoMetadataTuple;
86. DeliverToNull tonull1 = new DeliverToNull;
87. DeliverToNull tonull2 = new DeliverToNull;

88. |-provenanceRes-|=>metastoreex1.resource;
89. |-provenanceRes-|=>metastoreex2.resource;

90. ControlledRepeat rpl_2 = new ControlledRepeat;
91. ControlledRepeat rpl_3 = new ControlledRepeat;
92. DeliverToNull tonull1 = new DeliverToNull;
93. WaveformFilter filter = new WaveformFilter;
94. filter@gateway=preprocGateway;
95. WaveformDecimation decimation = new WaveformDecimation;
96. decimation@gateway=preprocGateway;
97. |-"filtertype=bandpass","minfrequency=0.05","maxfrequency=1","corners=1.0","zerophase=Tr
98. rpl_2.output=>filter.input;
99. rpl_2.repeatedOutput=>filter.parameters;
100. filter.output =>rpl_3.input;

101. |"factor=10"-|=>rpl_3.repeatedInput;
102. rpl_3.output=>decimation.input;
103. rpl_3.repeatedOutput=>decimation.parameters;
104. DeliverToNull nnull= new DeliverToNull;

105. filter.metadata=>metastoreex1.metastring;
106. decimation.metadata=>metastoreex2.metastring;
107. metastoreex1.datasetid=>metastoreex2.stepbackid;
108. return SeismoStage(<Connection input=rpl_2.input;Connection stepbackid=metastoreex1.stepbackid> => <Connection datasetid=metastoreex2.datasetid;Connection output=decimation.output;Connection metadata=metastoreex2.processedMetadata>);

109. };

110. PE<SeismoStage> whiten(String preprocGateway,String dataset,String provenanceRes){

111. SeismoMetadataTuple metastoreex1 = new SeismoMetadataTuple;
112. |
113. WaveformWhiten whiten=new WaveformWhiten;
114. whiten@gateway=preprocGateway;
115. DeliverToNull nnull= new DeliverToNull;
116. whiten.metadata=>metastoreex1.metastring;

117. return SeismoStage(<Connection input=whiten.input;Connection stepbackid=metastoreex1.stepbackid> =><Connection datasetid=metastoreex1.datasetid;Connection output=whiten.output;Connection metadata=metastoreex1.processedMetadata>);

118. };

119. //Stores the preprocessed datasets
120. PE (<Connection input=Connection stepbackid>=><Connection output>)

121. return PE(<Connection input=preprocstore.input;Connection stepbackid=metastoreex1.stepbackid> =><Connection output=preprocstore.output>);
use uk.org.ogsadai.ListRemove;
use uk.org.ogsadai.GenericActivity;
use uk.org.ogsadai.GenericTupleTransform;
use uk.org.ogsadai.TupleArithmeticProject;
use eu.admire.BuildClassifier;
use eu.admire.Classify;
use uk.org.ogsadai.TupleProjectByIds;
use uk.org.ogsadai.TupleSimpleMerge;
use uk.org.ogsadai.TupleArithmeticSample;
use uk.org.ogsadai.TupleUnionAll;
use uk.org.ogsadai.ListHead;
use uk.org.ogsadai.TuplePrintConfusionMatrix;

// SQL query

String expressionUKIDSS = "
SELECT
  x.slaveObjID AS sdss_id,
  x.masterObjID AS ukidss_id,
  l.yAperMag3-l.aY AS sd,
  l.j_1AperMag3-l.aJ AS ad,
  l.hAperMag3-l.aH AS qd,
  l.kAperMag3-l.aK AS wd
FROM lasSource AS l, lasSourceXDR5PhotoObj AS x
WHERE
  l.dec >= $REPLACE(lower)$ and
  l.dec < $REPLACE(upper)$ and
  x.masterObjID=l.sourceID AND
  l.yAperMag3 >0 and
  l.j_1AperMag3 >0 and
  l.hAperMag3 > 0 and
  l.kAperMag3 >0 and
  x.distanceMins<0.033333 AND
  x.sdssPrimary=1 AND
  x.distanceMins IN ( SELECT MIN(xx.distanceMins) FROM lasSourceXDR5PhotoObj as xx WHERE
    xx.masterObjID=x.masterObjID AND
    xx.sdssPrimary=1 )
";

String expressionSDSS = "
select p.objID ,
  (p.psfMag_u - p.extinction_u) as u,
  (p.psfMag_g - p.extinction_g) as g,
  (p.psfMag_r - p.extinction_r) as r,
  (p.psfMag_i - p.extinction_i) as i,
  (p.psfMag_z - p.extinction_z) as z,
  q.specobjid
From photoobjall as p LEFT JOIN dr5quasarcatalog as q ON p.specobjid=q.specobjid
WHERE
  p.objid IN ( $REPLACE(ids)$ )
  AND
  (p.type=6) AND
  (p.mode=1) AND
  (p.camcol<2 or p.colc<1383 or p.colc>1387) AND
  (p.camcol>5 or p.colc<1019 or p.colc>1031) AND
  (p.psfMag_g>14.5) AND
  (p.psfMag_g - p.extinction_g) <21) And
  (p.psfMagErr_u<=0.2 and p.psfMagErr_g<=0.2 and p.psfMagErr_r<=0.2 and p.psfMagErr_i<=0.2 and
  p.psfMagErr_z<=0.2) and
((p.psfMag_u - p.extinction_u) - (p.psfMag_g - p.extinction_g) < 1.0)
"

String script = "
include_class Java::uk.org.ogsadai.activity.generic.TupleHelper
include_class Java::uk.org.ogsadai.tuple.TupleTypes
def process(inputs, outputs)
    colA = TupleHelper.createColumnMetadata("lower", TupleTypes._DOUBLE)
colB = TupleHelper.createColumnMetadata("upper", TupleTypes._DOUBLE)
metadata = TupleHelper.createTupleMetadata([colA, colB].to_java)
writer = outputs.get("output")
writer.write(Java::uk.org.ogsadai.activity.io.ControlBlock::LIST_BEGIN);
writer.write(Java::uk.org.ogsadai.metadata.MetadataWrapper.new(metadata))
min_reader = inputs.get("min")
min = min_reader.read
max_reader = inputs.get("max")
max = max_reader.read
step_reader = inputs.get("step")
step = step_reader.read
value = min
while value+step <= max
    tuple = TupleHelper.createTuple([ value, value+step ].to_java)
    writer.write(tuple)
    value = value+step
end
if (value < max)
    tuple = TupleHelper.createTuple([ value, max ].to_java)
    writer.write(tuple)
end
writer.write(Java::uk.org.ogsadai.activity.io.ControlBlock::LIST_END);
end
"

String scriptToAddListMarkers = "
def process(inputs, outputs)
    writer = outputs.get("output")
    writer.write(Java::uk.org.ogsadai.activity.io.ControlBlock::LIST_BEGIN);
    reader = inputs.get("input")
    block = reader.read
    while block != Java::uk.org.ogsadai.activity.io.ControlBlock::NO_MORE_DATA
        writer.write(block)
        block = reader.read
    end
    writer.write(Java::uk.org.ogsadai.activity.io.ControlBlock::LIST_END);
end
"

GenericActivity generic = new GenericActivity;
// dec range: -24.18497619 -> 84.9799928869159
|- script | => generic.script;
|- 0-24.2 | => generic.min;
|- 85 | => generic.max;
|- 0.025 | => generic.step;
|- "jruby" | => generic.language;

Integer parallelUnits = 10;
// Round robin distribution of the ranges to the parallel branches
TupleArithmeticSample sample = new TupleArithmeticSample;
|- "Mod(Counter(), " + parallelUnits + ")" => sample.expression;
generic.output => sample.data;

// Union used to merge the results of the parallel branches
TupleUnionAll union = new TupleUnionAll;

// Create the parallel branches
for (Integer i = 0; i<parallelUnits; i++)
{
    StringReplace ukidssReplace = new StringReplace;
    |- repeat enough of expressionUKIDSS -| => ukidssReplace.template;
    sample.result[i] => ukidssReplace.data;

    SQLQuery query = new SQLQuery;
    ukidssReplace.result => query.expression;
    |- "UKIDSS-DR3-TAP" -| => query.resource;

    // Split into lots of small chunks
    ListSplit split = new ListSplit;
    |- repeat enough of 250 -| => split.size;
    query.data => split.data;

    GroupBy group = new GroupBy;
    split.result => group.data;
    |- repeat enough of [] -| => group.columnIds;
    |- repeat enough of \"STRING_AGGREGATE(sdss_id)\" -| => group.aggregates;
    |- repeat enough of \"ids\" -| => group.resultColumnNames;

    StringReplace replace = new StringReplace;
    |- repeat enough of expressionSDSS -| => replace.template;
    group.result => replace.data;

    SQLQuery sdssQuery = new SQLQuery;
    replace.result => sdssQuery.expression;
    |- "SDSS-DR7-TAP" -| => sdssQuery.resource;

    Buffer buffer = new Buffer;
    split.result => buffer.input;

    TupleThetaJoin join = new TupleThetaJoin;
    buffer.output => join.data1;
    sdssQuery.data => join.data2;
    |- repeat enough of \"sdss_id = objID\" -| => join.condition;
    |- repeat enough of \"data1\" -| => join.readFirst;

    GenericActivity addListMarkers = new GenericActivity;
    |- scriptToAddListMarkers -| => addListMarkers.script;
    |- \"jruby\" -| => addListMarkers.language;
    join.result => addListMarkers.input;

    ListRemove listRemove = new ListRemove;
    |- 2 -| => listRemove.level;
    addListMarkers.output => listRemove.input;

    TupleArithmeticProject project = new TupleArithmeticProject;
|- [ "u", "g", "r", "i", "z", "sd", "ad", "qd", "wd", "ReplaceNull(specObjId)" ] -| => project.expressions;
|- [ "u", "g", "r", "i", "z", "sd", "ad", "qd", "wd", "isQuasar" ] -| => project.resultColumnNames;
listRemove.output => project.data;

// Split the data, sample it and then combine
TupleArithmeticSample separateQuasars = new TupleArithmeticSample;
|- "isQuasar" -| => separateQuasars.expression;
project.result => separateQuasars.data;

TupleArithmeticSample sampleNonQuasars = new TupleArithmeticSample;
|- "Mod(Counter(), 29)" -| => sampleNonQuasars.expression; // 1/29 = 3.4% (roughly matches the quasar rate)
seperateQuasars.result[0] => sampleNonQuasars.data;

TupleUnionAll combineQuasarsAndNot = new TupleUnionAll;
sampleNonQuasars.result[0] => combineQuasarsAndNot.data[0];
seperateQuasars.result[1] => combineQuasarsAndNot.data[1];

combineQuasarsAndNot.data => union.data[i];

// Build classifier
BuildClassifier buildClassifier = new BuildClassifier;
|- "" -| => buildClassifier.options;
|- "weka.classifiers.bayes.NaiveBayes" -| => buildClassifier.algorithmClass;
head.head => buildClassifier.data;

//nominal columns
|- [ 9 ] -| => buildClassifier.columnIndices ; //Just the quasar/not quasar column
//nominal values
|- [ [ "0", "1" ] ] -| => buildClassifier.nominalValues;
//Indice to predict (isQuasar column)
|- 9 -| => buildClassifier.classIndex ;

// Classify the tail output
Classify classify = new Classify;
buildClassifier.classifier => classify.classifier;
head.tail => classify.data;

TupleProjectByIds projectQuasar = new TupleProjectByIds;
head.tail => projectQuasar.data;
|- ["isQuasar"] -| => projectQuasar.columnIds;

TupleSimpleMerge merge = new TupleSimpleMerge;
projectQuasar.result => merge.data[0];
classify.result => merge.data[1];
|- "predicted" -| => classify.resultColumnNames;

GroupBy confMatrixGroupBy = new GroupBy;
merge.result => confMatrixGroupBy.data;
|- [ "isQuasar", "predicted"] -| => confMatrixGroupBy.columnIds;
ADMIRE-REPORT ON PILOT APPLICATIONS DEPLOYMENT AND PLATFORM EVALUATION D6.6

DISPEL DOCUMENTS FOR APPLICATION SCENARIOS 89
Results fiveFeaturesResult = new Results;
drawConfMatrix5.result => fiveFeaturesResult.input;
|- "UsingFiveAttributes" -| => fiveFeaturesResult.name;

submit generic;
}