






The HUMBOLDT Workflow Editor

Background & User Manual 1.1 (August 2010)

Content:

Introduction & Background	2
Some Theoretical Background	2
Data-Dependency between nodes in a Workflow	2
Execution.....	3
Workflow Design: the Views & Graphical Elements	4
The Editor View	5
Processing Nodes	5
The Properties View	7
The Outline View	7
The Palette	8
Insert Data Sources.....	8
Insert Processing Nodes	9
Specify Dataflow	10
Deleting Elements	11
The Toolbar	11
Store 	11
Print 	12
Configure Workflow Editor 	12
Execute 	12
Validate Workflow 	12
Additional Tools.....	13
Execution & Scheduling.....	14
Execution Settings.....	14
Execution Results	15
Outlook.....	18

Introduction & Background

The Workflow Editor developed in the HUMBOLDT Project is a lightweight graphical user interface for geoprocessing workflow composition. It allows user to graphically compose OGC-conformant *Web Feature* and *Web Processing Services* into Workflows or dataflow graphs, which can directly be executed in the editor.

Some Theoretical Background

The geoprocessing workflows to be built with the Workflow Editor follow the structure of dataflow graphs, a well-know concept in software engineering. A dataflow graph is a graph $G = (V, E)$ with $V = G \cup D$ being a set of nodes and $E = (D \times G \cup G \times G)$ a set of directed edges. G is the set of *processing nodes* associated with computational components, in case of the workflow editor, geoprocessing components. D is the set of *data-providing nodes* associated with data providing components. A directed edge $e \in E$ connects a node associated with a data providing or computational component to another computational component. In case, a computational component has multiple inputs (often called input ports), there can be multiple edges pointing to it, each one associated with a different input. In case, a processing node has multiple outputs or the output shall be pushed as input to multiple input ports of a single or multiple other processing nodes, several edges from such a processing node might exist. The computational components represented by the nodes G in a dataflow graph are required to be function-like in the sense that they generate the same output given the same input and do not depend on some changing states (such as a global variable that is not a direct parameter). Further, the direct edges represent the dataflow between nodes. A node in a dataflow graph can “execute”, if the data of all other input nodes that provide the input via a dataflow edge is available. Due to the function-like nature of the processing nodes, the outcome of a whole dataflow graph of such components is determined solely by the input passed to it and can therefore be treated similar as a simple or atomic node and composed.

Data-Dependency between nodes in a Workflow

We say that a processing node $A \in G$ *depends* on a processing node $B \in G$ within a workflow or data flow graph if (and only if) there exists a directed path from B to A within the graph. Hence, a processing component node can execute, if all processing nodes it depends on have been executed successfully previously. In case, one processing node depends on another, we also say, they have a *data dependency*. Two way or cyclic dependencies (cyclic paths in the graph) between any two nodes are disallowed, since they result in a deadlock situation, where two processing nodes wait for each other to execute.

In the current implementation, the set of nodes D represent data providing **OGC Web Feature Services**. The set of nodes G represent data processing **OGC Web Processing Services**. OGC WFS serve as input nodes in a dataflow graph or workflow in the sense that they do not require any input except a simple invoke-me message (*GetFeature*) which can automatically be built by the system (the Workflow Execution Engine) based on the WFS URL and name of the Feature Type to be retrieved (we do not consider any WFS queries yet and simply assume that all instances of a particular Feature Type shall be retrieved). The OGC Web Processing Services require client defined input. Within a workflow, such input is either derived from another WPS (in case, a dataflow edge connects two WPS), from a WFS

(in case a dataflow edge connects a WFS to the WPS) or directly defined by the user when building the dataflow graph (e.g. by specifying a buffer distance as input to a process). The first two cases are represented by edges in the dataflow graph. Such inputs are called *complex*, following the OGC WPS specification. In the third case, we speak of *literal inputs* and refer to simple datatypes such as string, integer, double or any other simple value that can directly be provided by the composer of the dataflow graph or workflow.¹ Literal inputs are not represented by edges in the graph, but, as can be seen in **Figure 1**, are directly displayed within the processing nodes.

Execution

Due to the function like nature of the geoprocessing services, the concrete order of node execution, often called the schedule of the workflow, does not have an effect on the output data. Precisely, if two nodes $A \in V$ and $B \in V$ in a dataflow graph do **not** have a data dependency (if A does **not depend** on B or vice versa), it does not matter, which of them is executed first. As an example, consider a workflow shown in **Figure 1**.

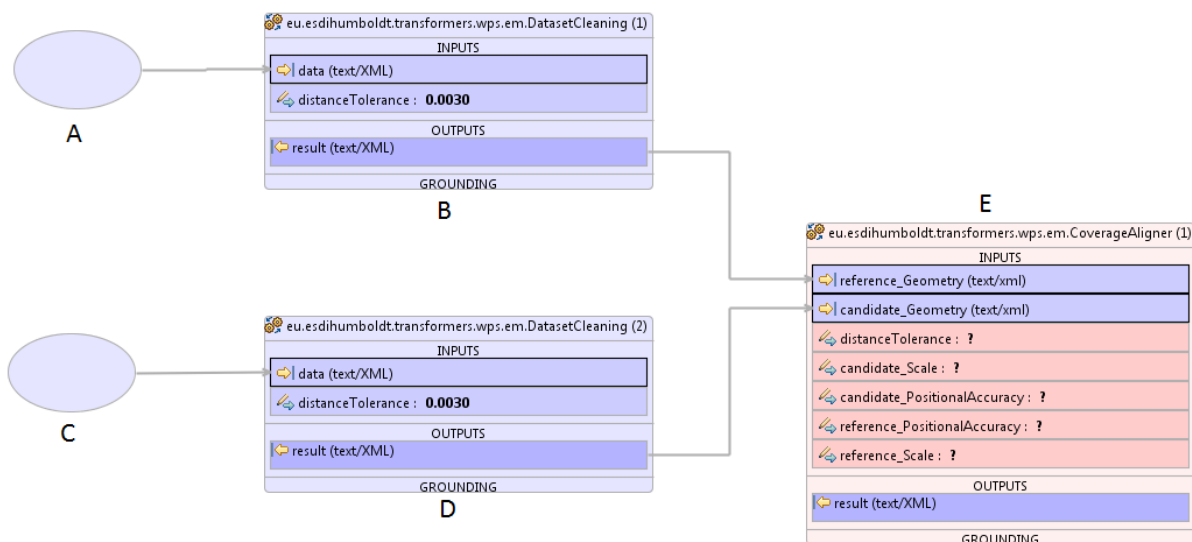


Figure 1: A simple workflow / data flow graph

Node B *depends* on node A, node D *depends* on node C since there is direct path from A to B and from C to D. Further, node E depends on A,B,C and D, since from each of them, there exists a directed path to E. However, A and B neither depend on C and D nor vice versa. Therefore, for the output of node E, it does not matter whether the sub-graph A-B is executed before or after the sub-graph C-D (and vice versa), they can even be executed in parallel. The output data of E does not change depending on the concrete execution order of the sub-graphs A-B or C-D. One big advantage of simple dataflow graph-based workflow models like this is that the human workflow composer does not have to be bothered with such execution-related issues such as parallelism. Those details can be handled automatically by the system.

¹ Note: the concrete distinction between complex and literal depends on the service signatures of the WPS to be composed with the Workflow Editor. The workflow editor simply parses the WPS ProcessDescriptions and hence simply follows the characterisation of literal vs. complex of the service providers building the process descriptions.

Workflow Design: the Views & Graphical Elements

The Workflow Editor consists of three main views, as shown in Figure 2:

- The main editor view containing a graphical representation of the processing and data source nodes and the dataflow interactions between them.
- The outline view containing an overview of the main editor view
- The properties view containing the properties of selected elements, namely Processing Nodes or Data Source Nodes.

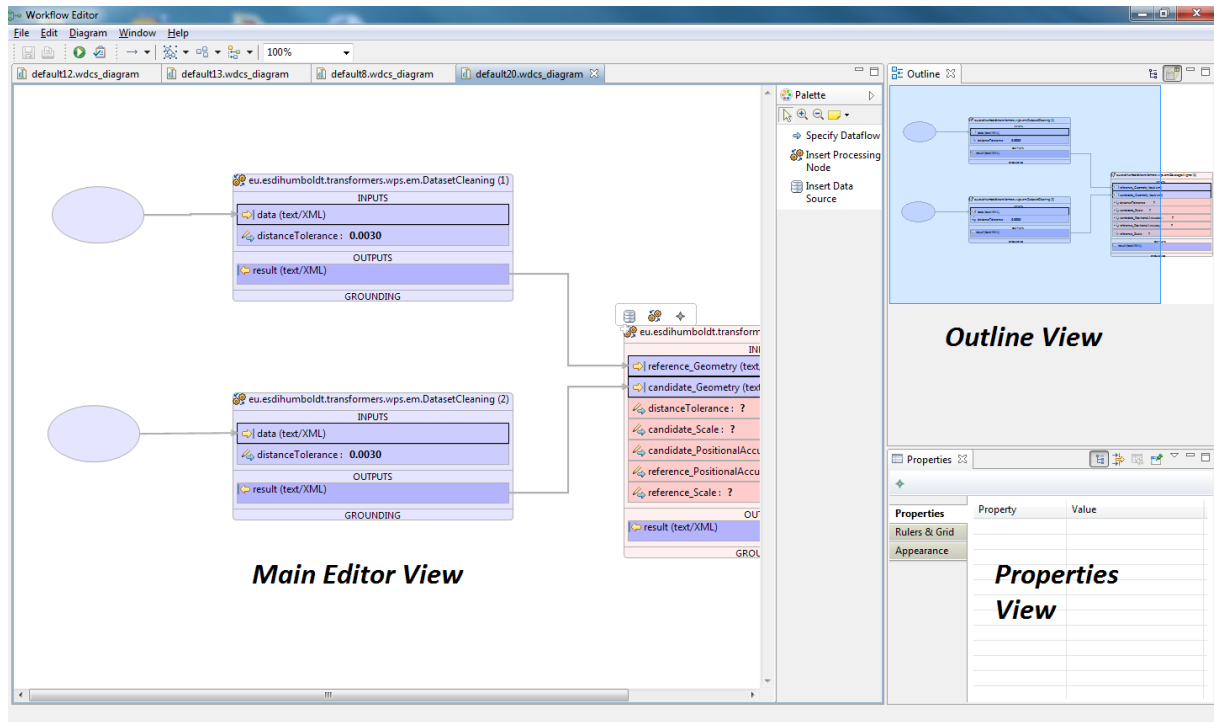


Figure 2: The main views

The individual views are described in detail in the following.

The Editor View

The main editor view holds the workflow graph, as shown in Figure 3. The elements of such a graph are data source nodes, dataflow connections and processing nodes.

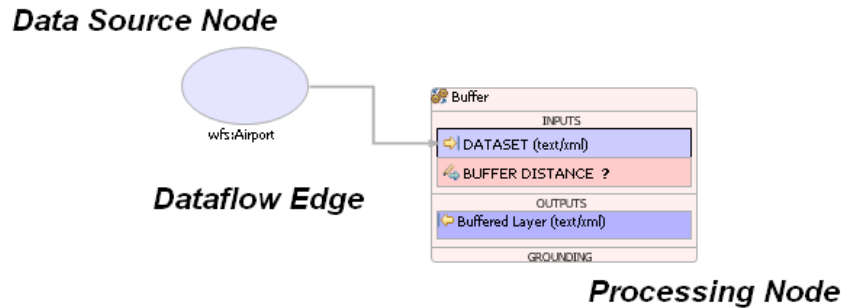




Figure 3: The editor view

The data source nodes are represented as ellipses and labelled with a feature type name. The dataflow connections are directed and indicate the dataflow during execution.

Processing Nodes

The processing nodes consist of three main elements, INPUTS, OUTPUTS and GROUNDING. The information contained within these sections is automatically parsed from WPS Process Descriptions.

The INPUT Section

The INPUTS section displays the names and types of the inputs, as indicated in the *WPS DescribeProcess* response for that process. An input can either be literal (indicated by ) or complex (indicated by ). Literal inputs can directly be typed in by users and therefore, double clicking a literal input display a dialogue that allows users to type in a value. Depending on whether the input value must be chosen from a predefined set of allowed input values (as indicated in the WPS Process Description) or not, the dialog differs. Figure 4 shows the simple dialogue allowing users to manually type in input values. When clicking the OK button, the system performs a simple type matching of the entered input value.

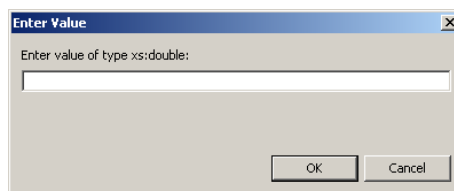


Figure 4: Enter Input Value Dialog

Figure 5 shows the dialog that allows users to select an input value out of a list of predefined allowed input values.

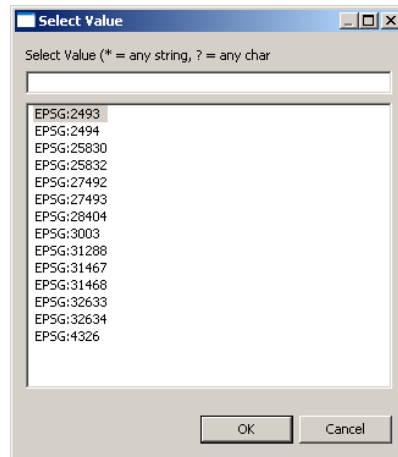


Figure 5: Select Input Value Dialog

Inputs can be satisfied or unsatisfied, indicated by *blue* resp. *red* colour (plus a question mark ?) in the editor view. A literal input is satisfied, if inputs have been specified for it. For example, if a literal input requires at least (minimum cardinality) and at most (maximum cardinality) one value specified for it, it is satisfied if there is exactly one value specified that instantiates the literal input when executing a workflow. If the input is optional, i.e. the minimum cardinality is zero, it is satisfied (and hence represented in blue), even if there is no value specified. Often, a default value is already specified for a literal input in the WPS *ProcessDescriptions* and hence, it is already retrieved in a satisfied state from the WPS. The same holds for a complex input, except that there must be a dataflow edge pointing to that input.

In Figure 3, the complex input of the buffer process is satisfied (as indicated by the blue colour), since a dataflow edge points to it. The literal input representing the buffer distance has not been specified yet and is therefore unsatisfied.

The OUTPUT Section

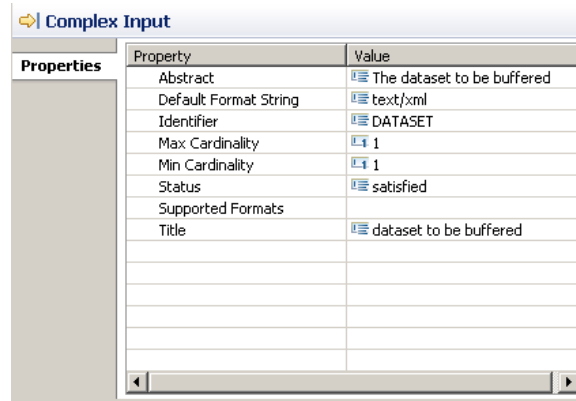
The output section contains the output parameters of the processing node. These can again be complex or literal.

The GROUNDING Section

The grounding section holds information on where the processing node is actually accessible. For WPS, this is the URL and the *ProcessIdentifier* of the process, as it appears in the WPS *GetCapabilities*- or *DescribeProcess* responses. By default, this section is minimized.

The Properties View

The properties view gives additional information on the elements that is not included in the main editor view.



Property	Value
Abstract	The dataset to be buffered
Default Format String	text/xml
Identifier	DATASET
Max Cardinality	1
Min Cardinality	1
Status	satisfied
Supported Formats	
Title	dataset to be buffered

Figure 6: The Properties View

Figure 6 gives an example of how the properties view looks like for a complex input.

The Outline View

There are two types of outline available, a tree view and an overview of the workflow diagram.

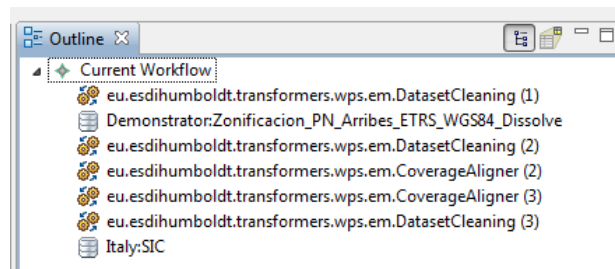


Figure 7: The outline tree view

The tree view (**Figure 1**) contains the workflow as top-level element and all nodes (processing and data source) as children.

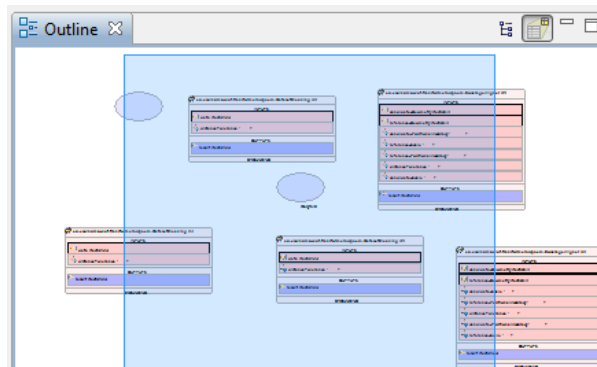


Figure 8: The outline overview

The overview (Figure 8) contains an overview of the whole workflow diagram (the blue box indicates the part currently displayed in the main editor view). Both views are synchronized with the main editor view. For example, selecting a node in the tree view results in a pan and

zoom to that particular node in the main editor view or panning the blue box in the overview results in a panning of the main editor view.

The Palette

The palette, by default located on the right hand side of the main editor view allows inserting data source- and processing nodes, as well as to specify dataflow connections between nodes. Additionally, it includes a number of simple tools such as tools for selecting elements, zooming in and out and attaching notes to elements in the main editor view.

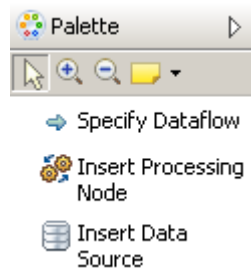


Figure 9: The Palette

Insert Data Sources

By clicking on the “*Insert Data Source*” item in the palette, the tool for inserting data sources gets selected. Clicking on the main editor view surface afterwards opens a simple data source discovery window, as shown in Figure 10.

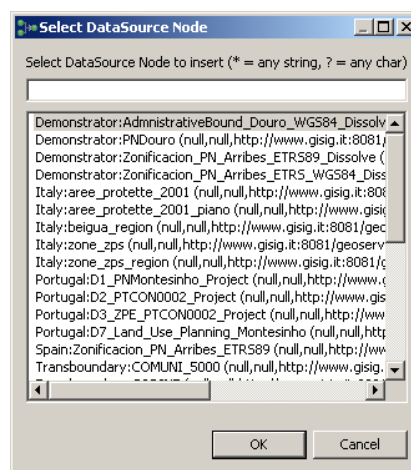


Figure 10: Data Source Insertion Window

By selecting one of the data sources and clicking the OK button, the selected data source gets inserted into the workflow and appears in the main editor view. The data sources displayed in this dialog are all data sources as registered to the Workflow Editor (see section *Configure Workflow Editor*). The data source insertion window allows a simple string search on the type name, spatial reference system, Bounding Box and URL of the data sources. For example, Figure 11 shows the example of searching for all data sources containing “Airpo” in their description.

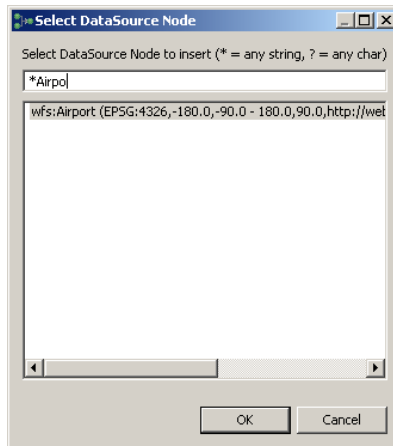


Figure 11: Searching for the type name Airport

Insert Processing Nodes

By clicking on the “Insert Processing Node” item in the palette, the tool for inserting processing nodes gets selected. Clicking on the main editor view surface afterwards opens a simple data source discovery window, as shown in Figure 12.

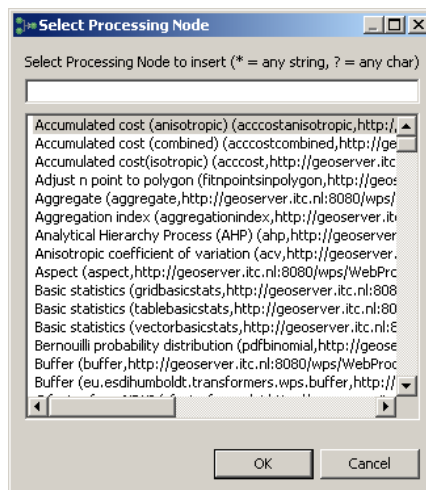


Figure 12: Processing Node Insertion Window

By selecting one of the processing nodes and clicking the OK button, the selected processing node gets inserted into the workflow and appears in the main editor view at the place, the user has clicked before. The processing nodes displayed in this dialog are all processing nodes as registered to the Workflow Editor (see section *Configure Workflow Editor*), i.e. all WPS Processes of all WPS registered.

The window contains a similar string-search functionality as for the data sources. For example Figure 13 shows a search for all processes containing the keyword “buffer” in their description.

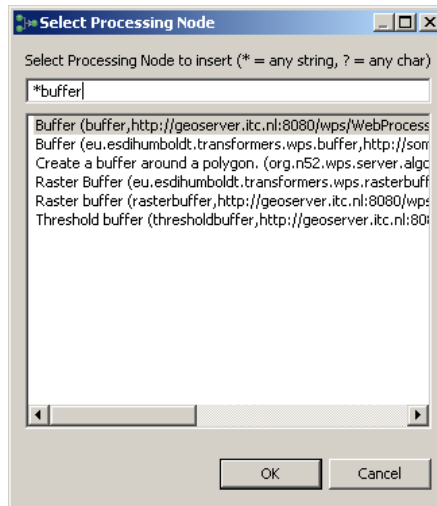


Figure 13: Searching for Buffer

Since multiple nodes of the same type can be inserted into a workflow (e.g. one single WPS Process can be inserted multiple times in different places of the workflow), each node receives a unique number that distinguishes it from other nodes with the same id. For example, in the workflow shown Figure 14, two times the process with the ID *eu.esdihumboldt.transformers.wps.em.DatasetCleaning* has been inserted.

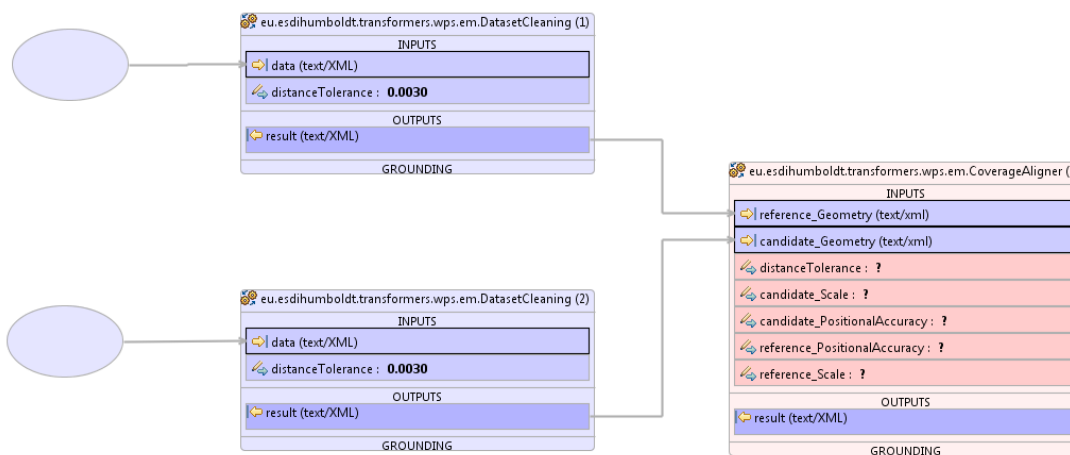


Figure 14: Two similar nodes in a workflow

In order to distinguish both processes, e.g. in the result report as well, they get an id unique to each individual node.

Specify Dataflow

By selecting the “*Specify Dataflow*” tool in the palette, users can specify the dataflow between nodes. A dataflow can be drawn from a data source node to the input of a processing node or from the output of a processing node to the input of another processing node. When connecting, the system performs a simple type matching strategy, described in the following section.

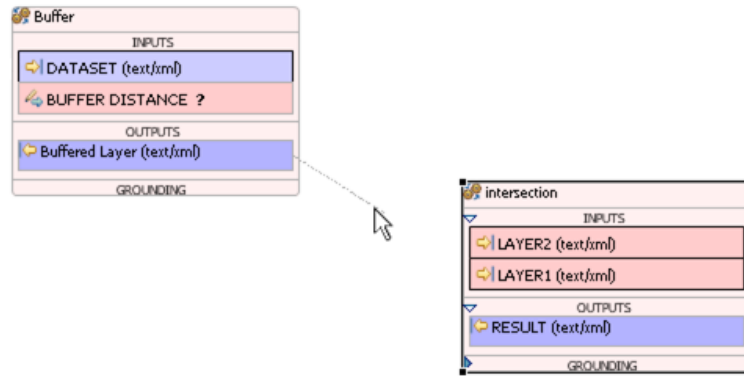


Figure 15: Create dataflow connection

Figure 15 gives an example of how to drag a dataflow connection from the output of a processing node to the input of another node.

Type Matching Strategy

The type matching is performed, when two nodes in the dataflow graph are about to be connected and relies on the metadata of two nodes (i.e. OGC WFS or OGC WPS metadata). However, since there might be errors in the metadata of the nodes, the outcome of the type matching is only given as a recommendation to the user and does not prevent her from connecting. In case, two WPS are about to be connected, the type matching strategy matches the supported formats of the two WPS, which usually consist of the allowed *MimeType*, *Schema* and *Encoding*.

In the future, we aim at extending this strategy by allowing service providers to formalise additional constraints or conditions on the input/output data that can not be specified in the metadata of current OGC WFS or OGC WPS Processes.

Deleting Elements

All elements on the diagram surface, i.e. data source nodes, processing nodes and dataflow edges can be deleted by selecting them and pressing the “del” button on the keyboard or opening the context menu with a right click on the element and selecting “Delete From Model”.

The Toolbar

The toolbar of the application offers some additional useful functionality (Figure 16).



Figure 16: The toolbar

Store

By clicking the store button, the workflow is stored to the file. The button is only enabled, if any changes (graphically by e.g. dragging nodes to different places or content wise by e.g. inserting new nodes or changing connections) have been made after the workflow has been saved the last time.

Print

The print button allows printing the workflow. In this version of the editor, printing functionality is not available and therefore, the button is always disabled.

Configure Workflow Editor

The configure workflow editor button opens the dialog shown in Figure 17.

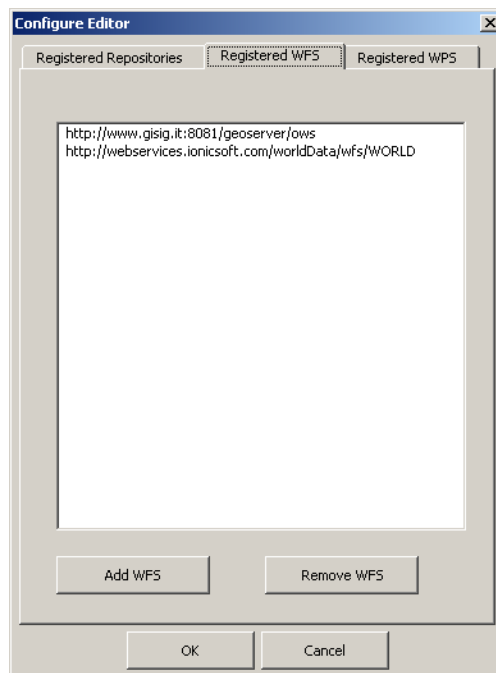


Figure 17: Configuring the Workflow Editor

The dialog allows to register OGC WFS and OGC WPS to the system, which can later on be composed in the editor.

Execute

The execute button allows executing the workflows. More details on workflow execution can be found in the relevant section.

Validate Workflow

The validate workflow button allows the validation of the current workflow opened in the editor. The validation strategy is twofold: First, it checks for unspecified inputs and hence inexecutable nodes in the workflow. An input is unspecified, if it is unclear where the value will come from when executing the workflow. Hence, a literal input is unspecified, if the workflow composer did not provide an input value. A complex input is unspecified, if there is no dataflow edge pointing to it. In the implementation, this input validation is a bit more complex and e.g. considers minimum and maximum cardinalities of the inputs as well.

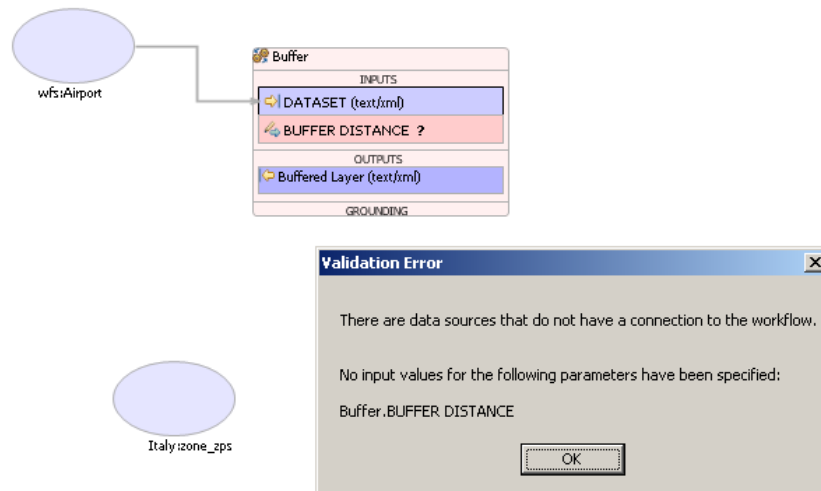


Figure 18: Workflow Validation

Further, the validation strategy performs some checks on the overall structure of the workflow (tree-like, unconnected nodes etc.). Figure 18 gives a small example of a validation error.

Additional Tools

There are additional tools in the toolbar that help the user in specifying the workflow. These include tools for zooming, changing the link appearance, arranging the graph automatically etc.

Execution & Scheduling

As described above, the output of a workflow or dataflow graph is solely determined by the data dependencies between nodes plus the input data. It is not determined by the concrete order (in time) of process execution, which is often called the *schedule* of the workflow. For data dependent nodes, the order of process execution is determined by the data dependency: If there is a directed path from a process A to a process B, B requires (directly or indirectly, depending on whether the path consists of a single edge or multiple) data processed by A and therefore **cannot** be executed before or in parallel to A. Hence, the schedule for data dependent nodes is fixed. However, for all other nodes that do not have a data dependency, the schedule is not fixed. Currently, the schedule for nodes that do not have a data dependency is determined by the order in which they have been inserted into the workflow by the human workflow designer. However, since the execution is usually done in parallel, this schedule order does not have much effect on e.g. things like runtime.

Execution Settings

By clicking the execution button on the toolbar (see previous section), a workflow validation is performed first. If there are no errors or if the user decides to proceed although there are errors, a dialog is shown that allows specifying execution settings for the workflow (Figure 19).

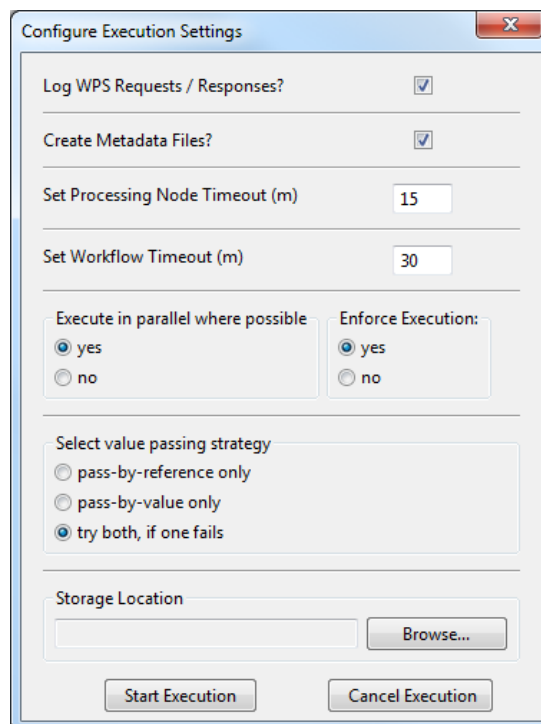


Figure 19: Execution Settings Dialog

The dialog allows configuring some basic execution settings for workflows.

Log WPS Requests / Responses: If enabled, the engine logs the WPS execute requests and responses. Inspecting the request / response messages might be helpful, in case some of the nodes in the workflow fail.

Create Metadata Files: If enabled, a metadata file will be created in the same folder and with the same name as the output files, containing basic information on the outputs, such as time of creation, name of processing node that created the result etc.

Set Processing Node Timeout: Sets the timeout in minutes for a single processing node in the workflow. Since the nodes are web services, it is recommended not to set the timeout to low, e.g. < 3 minutes.

Set Workflow Timeout: Sets the timeout in minutes for the whole workflow (must always be larger than the processing node timeout).

Execute in Parallel where possible: If set to *yes*, the execution engine will try to execute those processing nodes that do not have a data dependency in parallel (see introductory section). It is recommended to disable this setting only, if there are problems with the parallel execution, e.g. in case all processes in the workflow are hosted by a single WPS service and executing some in parallel results in a server overload.

Enforce Execution: If enabled, the execution engine will try to execute each processing node in the workflow, even if some nodes have already failed. For example, there might be two nodes or subgraphs in the workflow that do not have a data dependency and hence, each of them can successfully be executed, even if the other one fails. In case, the enforce execution setting is enabled, the execution engine will exactly try to do this: It tries to execute all nodes that do not have a data dependency on the nodes that have already failed. Hence, if enabled, the maximum number of possible (intermediate) results are retrieved. Therefore, it should only be disabled, if problems occur or if e.g. the workflow synchronises to a single target processing node and only the output of this node is of interest. In such case, this final node usually has a data dependency on each other node in the workflow and if no intermediate results are of interest, it makes sense to stop the execution if some node in the workflow fails. This is exactly what is done, if *enforce execution* is set to *false*;

Select value passing strategy: This setting allows to set the way, in which the remote WPS are executed.

pass-by-reference only: Forces the execution engine to pass the data by reference to only. Since some WPS might not support passing-by-reference, this setting should only be used, if *pass-by-value* fails.

pass-by-value only: Forces the execution engine to pass the data by value only. This setting is recommended, if the workflow e.g. only consists of WPS processing literal data such as integers, strings etc.

try-both: This is the recommended setting since it first tries passing the data by reference on each WPS first and if this does not succeed, it tries passing by value. However, if there are WPS in the workflow that e.g. do not allow passing data by reference, this setting might be slower in terms of runtime compared to the *pass-by-value only* setting.

Storage Location: Allows specifying the location on hard-drive, where the results shall be stored.

Execution Results

After the workflow has been executed, a dialog is shown that presents the results. The upper part/table of the dialog contains the list of valid results. The lower part / table contains invalid results. The dialog contains not only the end result of the workflow (e.g. the output of the last processing node) but all intermediate results as well. Further, a single line does not only correspond to a WPS but to the output of a WPS. For example, a successfully executed WPS with 5 outputs results in five lines / entries in the table. Therefore, in addition to the name and number of processing node that produced the result, the tables contain a column *output ID*, uniquely identifying the output, the result has been obtained from.

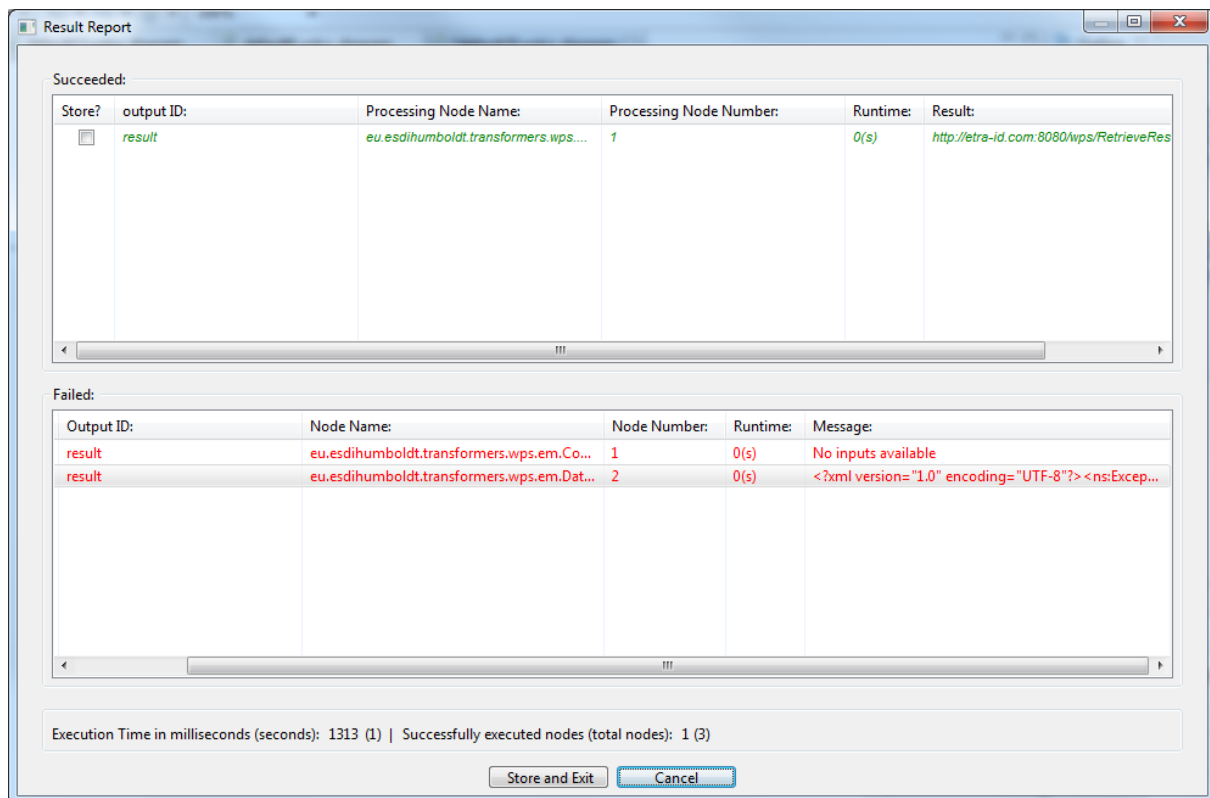


Figure 20: The Results Report Dialog

For those results obtained successfully, the user is allowed to specify whether he wants to store it on disk. This can be done by checking the check.box in the *Store?* column. Further, the entries contain information on runtime and a pointer to the result, in case the data has been passed by reference and the individual WPS stored the results on the service. In case, the data has been passed by value, the *Result* directly holds an extract of the data, e.g. some part of a GML file. However, inspecting the file is not possible in this dialog and requires storing it on disk.

The *Failed*-table additionally contains a *Message*-column holding different kinds of error information. For example, in case, a WPS service has been executed but has delivered an exception / error as response message, the column *Message* holds this response message directly obtained from the server. In case, the WPS has not been executed because some other WPS nodes it depended on have been failed, it contains a Message from the execution-engine saying “No inputs available”. The entries in the *Failed*-table can be double clicked in order to view the full error message / exception report, as shown exemplarily in Figure 21 for an error/exception response from a remote WPS.

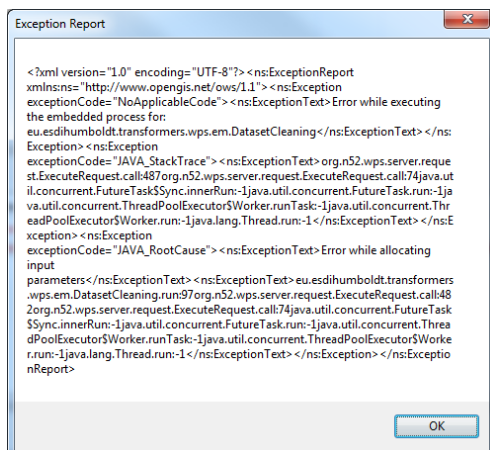


Figure 21: An exception report from a WPS

The entries in the *Succeeded*-table are displayed differently, depending on whether they represent a final output of the workflow (i.e. an output which is not pushed as input to another node, e.g. the output *result* of *eu.esdihumboldt.transformers.wps.em.DatasetCleaning* in Figure 22) or whether they represent intermediate results (i.e. those that have been used as input for another node, e.g. *outdata* of *complexVector* in Figure 22).

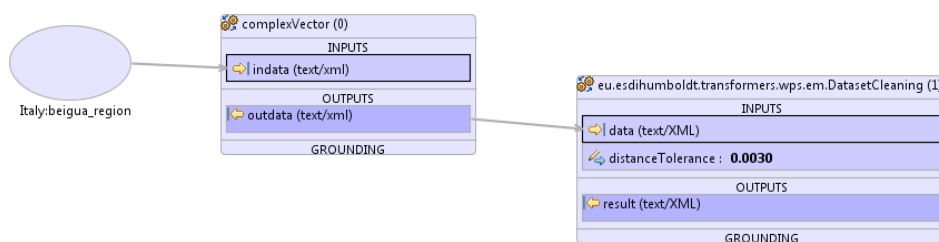


Figure 22: A simple workflow with one intermediate, one end result

For example, for the workflow shown in Figure 22, the entries in the result table appear as shown in Figure 23.

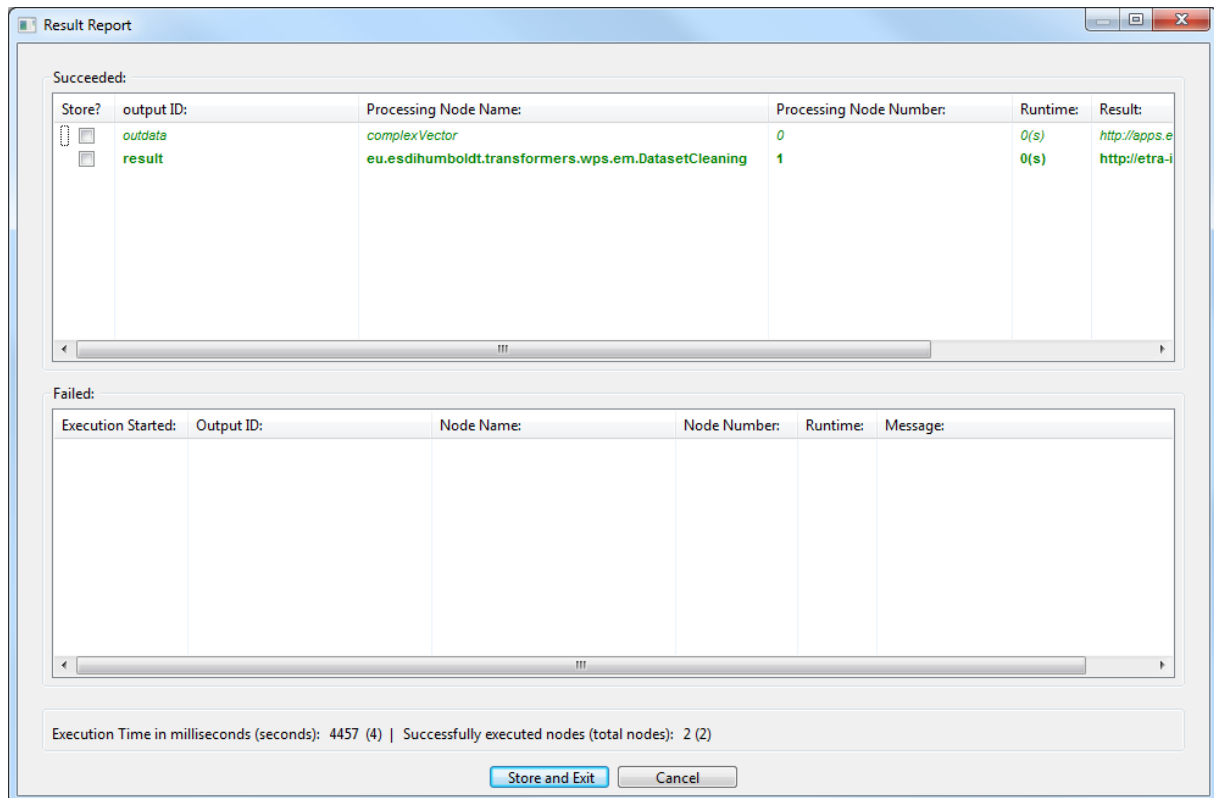


Figure 23: One intermediate, one final result

The entry representing the single output *outdata* of the process *complexVector* appears in italic letters, indicating that it represents an intermediate result. The entry representing the single output *result* of the process *eu.esdihumboldt.transformers.wps.em.DatasetCleaning* appears in bold and non-italic letters, indicating that it is a final result, which has not been pushed as input to another node.

Outlook

Some issues remain open that will be addressed in future version of the Workflow Editor. For example, we plan to allow users to insert whole workflows as nodes into another workflow, allowing the reuse of workflows and reducing the size and complexity of the workflow diagrams. Further, service providers shall be enabled to specify cross-parameter conditions when registering services, such as the information that the service they register outputs the same spatial reference system as the input. This would offer additional automated help in the composition process.