The Odysseus Operator Framework

This document describes the basic concepts of the Odysseus operator framework and shows how to extend the framework with new operators. In Odysseus an operator consumes events streams, does some calculations and produces results event streams. Different operators can be connected by a publish /subscribe concept to form an operator plan (or query plan).

Logical and physical operators

A logical operator can be compared to an algebra operator. It contains not the concrete algorithms for the processing of data but describes what to do with the data, e.g. in a projection it states the attributes to deliver, and in a join it gives the join predicates. A physical operator is one possible implementation of an algorithm to process the operation. So, different physical operators can be provided for the same logical operator. Logical operators are translated by transformation rules to physical operators.

PQL

A typical approach to describe queries is done by declarative query languages like SQL. Odysseus also provides a stream optimized version of SQL named CQL. Because data stream queries are often hard to express with Select-From-Where clauses, Odysseus also provides a more operator based query language named PQL. Within this language the logical operators are the basic building blocks. Further information to PQL can be found in the corresponding documentation The Odysseus Procedural Query Language (PQL) Framework.

Source, Sink and Pipes

Physical operators are divided into operators that add data to the systems (sources), remove data from the system (sinks) and process data (pipes). In a more query plan like view, the sources are operators with no input but with output port, the sinks are operators with only inputs ports and the pipes are both sources and sinks. Odysseus provides basic implementations for sources (AbstractSource), sinks (AbstractSinks) and pipes (AbstractPipe) that must be used to create own operators. The abstract implementations provide most necessary common processing code to concentrate on special processing for the concrete operator. In the following we will demonstrate the creation of a simple route operator, that routes data tuples to different output ports depending on some predicates. The physical operators are connected by a subscription.

Interaction

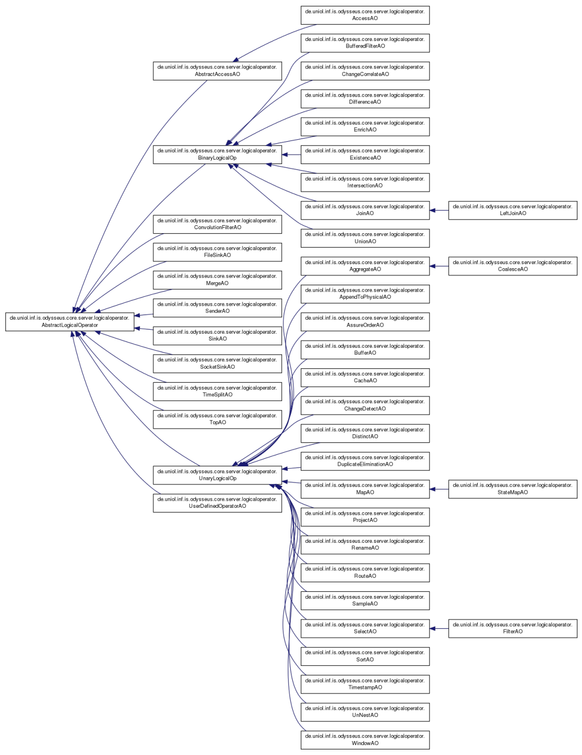
The Odysseus framework calls the operator. Odysseus implements the open - next - close Protocoll:

- open: Is called from the client (e.g. in the UI). This command initializes the operators from the top operators until the leafs.
- next:
- close:

```
TODO: Interaction between sources and sinks
```

```
 Open, next, close, transfer, process_open, process_next, process_close
```

Available Operators


de.uniol.inf.is.odysseus.core.server.logicaloperator.AbstractLogicalOperator
de.uniol.inf.is.odysseus.core.server.logicaloperator.AbstractAccessAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.BinaryLogicalOp
de.uniol.inf.is.odysseus.core.server.logicaloperator.ConvolutionFilterAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.FileSinkAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.MergeAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.SenderAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.SinkAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.SocketSinkAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.TimeSplitAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.TopAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.UnaryLogicalOp
de.uniol.inf.is.odysseus.core.server.logicaloperator.UserDefinedOperatorAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.AccessAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.BufferedFilterAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.ChangeCorrelateAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.DifferenceAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.EnrichAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.ExistenceAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.IntersectionAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.JoinAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.UnionAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.AggregateAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.AppendToPhysicalAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.AssureOrderAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.BufferAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.CacheAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.ChangeDetectAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.DistinctAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.DuplicateEliminationAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.MapAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.ProjectAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.RenameAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.RouteAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.SampleAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.SelectAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.SortAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.TimestampAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.UnNestAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.WindowAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.LeftJoinAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.CoalesceAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.StateMapAO
de.uniol.inf.is.odysseus.core.server.logicaloperator.FilterAO

User defined Operators

A rather simple way for the creation of user defined operators is to use the UDO-PQL Operator. For further information see PQL The Odysseus Procedural Query Language (PQL) Framework