Operator Definition Language (ODL)

This document describes the steps to create a new user defined operator with ODL.

Name

First you need to choose a meaningful name for the new operator. Note, there is no other operator with the same name.

Example

```
operator ODLSelect {
        //...
}
```

Metadata

Next you can set metadata to configure the operator with static information.

Example

```
operator ODLSelect(outputMode = "INPUT", minInputPorts = 1, maxInputPorts = 1){
    //...
}
```

The following table shows the available metadata:

Metadata	Possible values

outputMode (important)	• "INPUT"
	  ◦ read element wil not be modified
	• "MODIFIED_INPUT" (default)
	  ◦ read element will be modified
	• "NEW_ELEMENT"
	  ◦ operator creates a new element

persistent	• true ◦ operator will be automatically added to operator framework after each start of Odysseus • false (default)
minInputPorts	• Integer-Value (default 1) ◦ Minimal number of ports that the operator needs
maxInputPorts	• Integer-Value (default 2147483647) ◦ Maximum number of ports that the operator needs

Please look at the LogicalOperator-Interface of Odysseus for more available metadata (e.g. doc, url, category).

Parameter

You need to define parameters if the operator should be configurable while building a continuous query. A parameter can be optional and consists of a data type and a name.

Example

```
operator ODLSelect(outputMode = "INPUT", minInputPorts = 1, maxInputPorts = 1){

    parameter IPredicate predicate;

    optional parameter int heartbeatrate;

        //...
}
```

Note that not every data type can be used for a parameter because each data type must have a corresponding parameter class in Odysseus. The following table shows the available data types:

Data type	Parameter class
String	StringParameter FileNameParameter HttpStringParameter
Boolean	BooleanParameter
Byte	ByteParameter
Integer	IntegerParameter
Double	DoubleParameter
Long	LongParameter
Resource	ResourceParameter AccessAOSourceParameter
AggregateItem	AggregateItemParameter
BitVector	BitVectorParameter
RenameAttribute	CreateAndRenameSDFAttributeParameter
SDFAttribute	CreateSDFAttributeParameter ResolvedSDFAttributeParameter
File	ValidatedFileNameParameter
IMetaAttribute	MetaAttributeParameter
IPhysicalOperator	PhysicalOperatorParameter
IPredicate	PredicateParameter
NamedExpression	SDFExpressionParameter
AccessAO	SourceParameter
TimeValueItem	TimeParameter

Some data types have more than one corresponding parameter class. In these cases you can optionally set a parameter class as a metadata of a parameter. Please look at the Parameter-Interface of Odysseus for more available metadata.

Example

```
operator ODLProject(outputMode ="MODIFIED_INPUT", minInputPorts = 1, maxInputPorts = 1){

    parameter(type=ResolvedSDFAttributeParameter) SDFAttribute[] attributes;

        //...
}
```

Attributes

Besides parameters it is also possible to define attributes to store operator's data.

Example

```
operator ODLJoin(outputMode = "NEW_ELEMENT", minInputPorts = 2, maxInputPorts = 2){

    ITimeIntervalSweepArea[] areas;
    IDataMergeFunction dataMerge;
    ITransferArea transferFunction;
    IMetadataMergeFunction metadataMerge;

        //...
}
```

Validate methods

Each parameter can have a validate method. This method must return whether the value of the parameter is correct.

Example

```
operator ODLProject(outputMode ="MODIFIED_INPUT", minInputPorts = 1, maxInputPorts = 1){

     validate attributes {
        if (this.attributes.isEmpty) {
            error("attributes have to be set");
            return false;
        }
        return true;
    }

        //...
}
```

There is also a validate method which can be used if a parameter has dependencies to other parameters.

Example

```
operator ODLProject(outputMode ="MODIFIED_INPUT", minInputPorts = 1, maxInputPorts = 1){

     validate {
        return true;
    }

        //...
}
```

Event methods

These methods are automatically called when events occur. The processNext-method is the most important event method and is called whenever a new stream element is received. The type of a stream element can be Tuple, KeyValueObject or other subtypes of IStreamObject.

Example

```
operator ODLSelect(outputMode = "INPUT", minInputPorts = 1, maxInputPorts = 1){

    on processNext(Tuple tuple, int port) {
        if (predicate.evaluate(tuple)) {
            sendStreamElement(tuple);
        }
    }

    on processPunctuation(IPunctuation punctuation, int port) {
        sendPunctuation(punctuation);
    }

        //...
}
```

The following table shows an overview of available event methods. Note that you can call methods of AbstractLogicalOperator or AbstractPipe in event methods, e.g. to send stream elements or punctuations.

Event method	Description
processNext(IStreamObject object, int port)	New stream element is received
processPunctuation(IPunctuation punctuation, int port)	New punctuation is received
processOpen()	Processing has started
processClose()	Processing has stopped
processDone()	No more stream elements are sent
processDone(int port)	No more stream elements are sent from a source
createOutputSchema (int port)	Output schema has to be created
sourceSubscribed()	New source is subscribed
sourceUnsubscribed()	Source is unsubscribed
ao_init()	Logical operator is initialized
po_init()	Physical operator is initialized

AO & PO methods

Besides validate and event methods it is also possible to create normal methods. The ao-keyword should be used if a method needs to call a method of AbstractLogicalOperator. The po-keyword should be used if a method needs to call a method of AbstractPipe.

Example

```
getOtherPort(int port) : int {
        return (port+1)%2;
}

ao getBaseTimeUnit : TimeUnit {
        SDFSchema schema = getInputSchema(0);
        SDFConstraint constraint = schema.getConstraint(SDFConstraint::BASE_TIME_UNIT);
        return constraint.value;
}

po ping(IPunctuation punctuation) {
        sendPunctuation(punctuation);
}
```

Grammar

Grammar

```
ODLModel              ::= (Namespace)* (UserOperator | Class | Interface)*.
UserOperator          ::= "operator" ID ("(" (Metadata ("," Metadata))? ")")? "{" (ODLAttribute | ODLMethod)
"}".
ODLAttribute          ::= (("optional")? "parameter")? ("(" (Metadata ("," Metadata))* ")")? Attribute.
ODLMethod             ::= ("on" | "validate" | "override" | ("override")? "ao" | ("override")? "po")?
(MethodDeclaration)? StatementBlock.
```

Examples

ODLProject

```
operator ODLProject(outputMode ="MODIFIED_INPUT", minInputPorts = 1, maxInputPorts = 1){

    parameter(type=ResolvedSDFAttributeParameter) SDFAttribute[] attributes;

    int[] restrictList;

    on processOpen {
        if (restrictList == null) {
            restrictList = SDFSchema::calcRestrictList(getInputSchema(0), this.outputSchema);
        }
    }

    on createOutputSchema(int port) : SDFSchema{
        return SDFSchemaFactory::createNewWithAttributes(attributes, getInputSchema(0));
    }

    on processNext(Tuple tuple, int port) {
        Tuple out = tuple.restrict(restrictList, false);
        sendStreamElement(out);
    }

    on processPunctuation(IPunctuation punctuation, int port) {
        sendPunctuation(punctuation);
    }
}
```

ODLSelect

```
operator ODLSelect(outputMode = "INPUT", minInputPorts = 1, maxInputPorts = 1){

    parameter IPredicate predicate;

    on processOpen {
        predicate.init;
    }

    on processPunctuation(IPunctuation punctuation, int port) {
        sendPunctuation(punctuation);
    }

    on processNext(Tuple tuple, int port) {
        if (predicate.evaluate(tuple)) {
            sendStreamElement(tuple);
        }
    }
}
```

ODLJoin

```
operator ODLJoin(outputMode = "NEW_ELEMENT", minInputPorts = 2, maxInputPorts = 2){

    optional parameter IPredicate predicate = TruePredicate::getInstance();

    optional parameter(type = EnumParameter) Cardinalities card;

    ITimeIntervalSweepArea[] areas = [];

    IDataMergeFunction dataMerge;

    ITransferArea transferFunction;

    IMetadataMergeFunction metadataMerge;

    on po_init {
        String areaName = "TIJoinSA";
        areas[0] = SweepAreaRegistry::getSweepArea(areaName);
        areas[0].queryPredicate = predicate;

        areas[1] = SweepAreaRegistry::getSweepArea(areaName);
        areas[1].queryPredicate = predicate;
    }

    on processOpen {
        if (dataMerge == null) {
            dataMerge = new RelationalMergeFunction(this.outputSchema.size);
            transferFunction = new TITransferArea();
            List leftMeta = getInputSchema(0).metaAttributeNames;
            List rightMeta = getInputSchema(1).metaAttributeNames;
            metadataMerge = MetadataRegistry::getMergeFunction(leftMeta,rightMeta);
        }
        areas[0].clear;
        areas[1].clear;
        dataMerge.init;
        transferFunction.init(this, this.subscribedToSource.size);
        metadataMerge.init;
        predicate.init;
    }

    on createOutputSchema(int arg0) : SDFSchema{
        SDFSchema left = getSubscribedToSource(0).schema;
        SDFSchema right = getSubscribedToSource(1).schema;
        return SDFSchema::join(left, right);
    }
```

```
on processNext(Tuple tuple, int port) {
    transferFunction.newElement(tuple, port);
    int otherPort = getOtherPort(port);
    Order order = Order::fromOrdinal(port);
    areas[otherPort].purgeElements(tuple, order);
    boolean extract = false;
    if (card != null) {
        switch (card.toString) {
            case "ONE_ONE":
                extract = true;
                break;
            case "MANY_ONE":
                extract = port == 1;
                break;
            case "ONE_MANY":
                extract = port == 0;
                break;
            default:
                break;
        }
    }

    Iterator qualifies = areas[otherPort].queryCopy(tuple, order, extract);

    boolean hit = qualifies.hasNext;
    while(qualifies.hasNext) {
        Tuple next = qualifies.next;
        Tuple newElement = dataMerge.merge(tuple, next, metadataMerge, order);
        transferFunction.transfer(newElement);
    }

    if (card == null || card == Cardinalities::MANY_MANY) {
            areas[port].insert(tuple);
    } else {
        switch (card.toString) {
        case "ONE_ONE":
            if (!hit) {
                areas[port].insert(tuple);
            }
            break;
        case "ONE_MANY":
            if (port == 0 || (port == 1 && !hit)) {
                areas[port].insert(tuple);
            }
            break;
        case "MANY_ONE":
            if (port == 1 || (port == 0 && !hit)) {
                areas[port].insert(tuple);
            }
            break;
        default:
            areas[port].insert(tuple);
            break;
        }
    }
}

on processPunctuation(IPunctuation punctuation, int port) {
    if (punctuation.isHeartbeat) {
        areas[getOtherPort(port)].purgeElementsBefore(punctuation.time);
    }
    transferFunction.sendPunctuation(punctuation);
    transferFunction.newElement(punctuation, port);
}

on processDone(int port) {
    transferFunction.done(port);
}

po getOtherPort(int port) : int {
```

```
        if (port == 0) {
            return 1;
        } else {
            return 0;
        }
    }
}
```