Command Operator

Remark: Although part of the core system, this operator is still BETA!

This operator allow to evaluate expressions (similar to Map operator). The operator needs MEP Functions that return Command Objects. In the same way as map, the operator is triggered when an input event arrives. This operator can be used to create actors.

Such MEP functions can be found in the Command section.

Parameter:

- CommandExpression: The command expression that should be used. See examples at Command

Example

```
///Sensor management start logging
cmd = COMMAND({CommandExpression='startLogging("Camera1234"))'}, in)

/// change Timer period
cmd = COMMAND({commandExpression='setPeriod("MyTimer.transport", 1000.0))'}, in)

/// add query
cmd = COMMAND({CommandExpression='addQuery("data = ACCESS(...)", "PQL")'}, in)
```

Here are more complex example:

```
#PARSER PQL
#QNAME StopAllQueries

#ADDQUERY
planmodifications = ACCESS({
          source='PlanModificationWatcher',
          wrapper='GenericPush',
          transport='planmodificationwatcher',
          datahandler='Tuple'
        }
      )
sel = SELECT({PREDICATE = 'EVENTTYPE == "QUERY_START" AND queryID > 4'}, planmodifications)

cmd = COMMAND({CommandExpression='stopQuery(queryId))'}, sel)
```

```
#PARSER PQL
#ADDQUERY
timer = TIMER({
            id = 'timer1',
            period = 1000,
            starttime = 0 ,
            source = 'source'
          }
        )

filter = SELECT({
              predicate = 'time % 2 == 0',
              heartbeatrate = 1
            },
            timer
          )

command = COMMAND({
              COMMANDEXPRESSION = 'updateTransportOption("mqttsink.transport","TOPIC","TEST"+toString(time %
2))'
            },
            filter
          )

out = SENDER({
         transport = 'MQTT',
         protocol = 'csv',
         sink = 'mqttsink',
         WRAPPER = 'GenericPush',
         datahandler = 'tuple',
         options = [
           ['topic','TEST'],
           ['Broker','tcp://192.168.2.34:1888'],
           ['Client_ID','OdysseusSender']
         ]
       }, command
     )
```

```
#PARSER PQL
#ADDQUERY
timer = TIMER({id = 'timer1', PERIOD = 1000, STARTTIME = 0 , SOURCE = 'source'})

filter = SELECT({PREDICATE = 'time % 2 == 0'},timer)

command = COMMAND({COMMANDEXPRESSION = 'updateTransportOption("timer1.transport","PERIOD",toString((counter()+1)
*1000))'}, filter)
```

updateTransportOption can be used to change some options that are given for the Transport handler.

The first parameter defines the target, where to apply the method (same in other handlers). Here the part in front of the dot defines the id of the operator (here timer1) and the second part tells, that the transport handler ist the target. There is a similar function updateProtocolOption where you should use (<id>.protocol). So, timer1.transport means, that you want to access the transport handler of the operator with the id timer1.

The second parameter is the option that should be changed (here PERIOD) and the third one the new value as a string (here an increasing counter).

The commandexpression is always evaluted when an object arrives at the operator and the content of the object can be used in any parameter.

We are working on a version, that get activated with some specific punctuation.

Remark: To react on this requests, the transport handler needs to implement the optionsMapChanged(String key, String value) method. It is typically so, that the handler will not allow to change any value.

You could also use punctuations to force a command execution:

```
#ADDQUERY
timer = TIMER({
            id = 'timer1',
            period = 1000,
            starttime = 0 ,
            source = 'source'
          }
        )

filter = SELECT({
              predicate = 'time % 2 == 0',
              heartbeatrate = 1
            },
            timer
          )

command = COMMAND({
              SUPPRESSPUNCTUATIONREACTEDON = true,
              reactOnPunctuations = [['Heartbeat','updateTransportOption("timer1.transport","PERIOD",toString
(point))']]
            },
            filter
          )
```

- reactOnPunctuations: Contains a list of punctuations and expression for which a reaction should be forced
- SUPPRESSPUNCTUATIONREACTEDON: if set to true (default is false) the "used" punctuation will not be resend by the operator