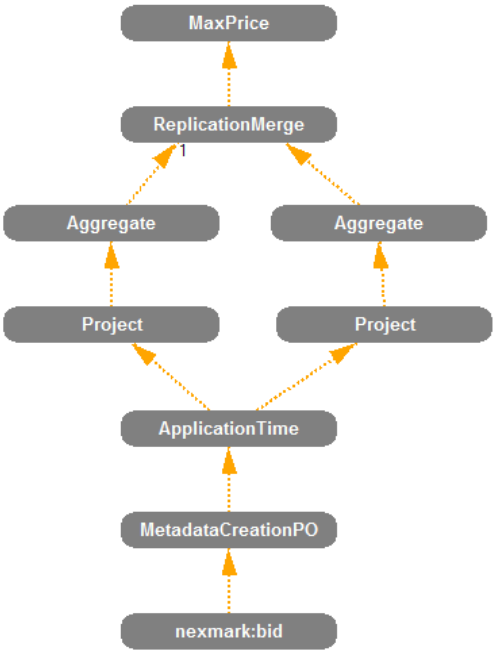
Replication

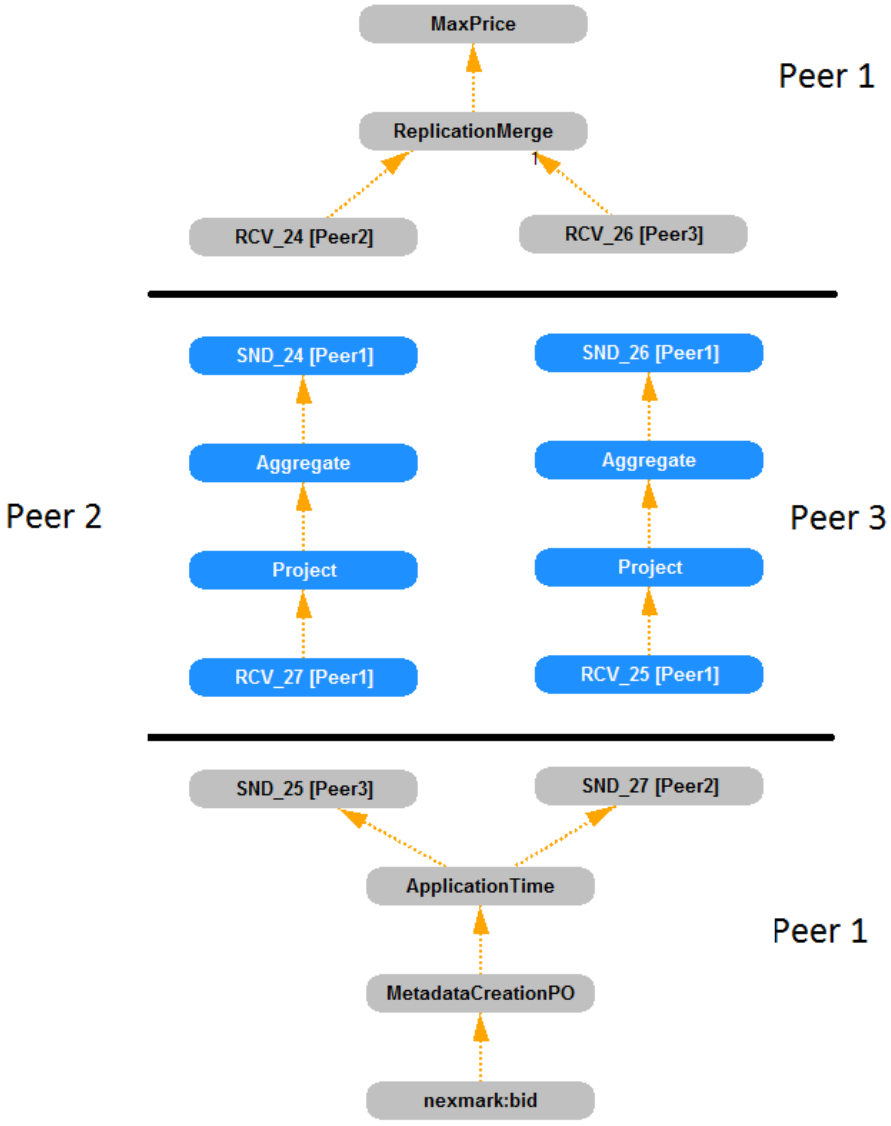
The replication mechanism in Odysseus

The replication mechanism is part of the *Peer-Feature* and can increase the reliability of a distributed query processing in a Peer-to-Peer network of Odysseus instances. Figure 1 shows a simplified operator graph. All operators between sources and sinks are replicated several times. Each replica gets the complete data stream from its sources for processing. Without failures, each replica produces the same result stream. To merge the different result streams, a `ReplicationMerge` operator is inserted to eliminate duplicates.



A simplified operator graph for replication.

Typically, replicas are to be executed on different peers. If one replica (peer) fails, there is at least one other replica processing the same data stream and the query still provides output streams. A normal, not simplified operator graph for the same query as in figure 1 is shown in figure 2. Here, special `JxtaSender` and `JxtaReceiver` operators handle the transport of data stream elements from one peer to another.



A normal, not simplified operator graph for replication.

A query partitioning provides the basis for the replication mechanism. The partitioning of a query means the creation of disjunctive partial queries with the objective to determine those operators, which shall be executed on the same peer. In figure 1 and figure 2 a partition strategy was used, which creates one single partial query with all operators. If the user decides to choose another partition strategy resulting in more than one partial query, the operator graph for the replication changes. This is shown in figure 3. Here, each operator builds its own partial query. The replication mechanism merges after each of those partial queries. The result is that the query will produce results as long as at least one replica of each partial query remains.

A normal, not simplified operator graph with multiple mergers for replication.

The activation of replication via Odysseus Script

The user can enable the replication mechanism via Odysseus Script. Listing 1 shows the query definition in PQL (with Odysseus Script) for the query shown in figure 2. At first, the query distribution has to be enabled. After that, a series of distribution commands follow: The partition strategy defines the partial queries as described above (*QueryCloud* creates one partial query containing all operators). The modification command enables the replication. The additional argument is the degree of replication (how often shall the query be executed in the network). The last distribution command (allocation) defines which peer executes which partial query (here, *roundrobin* is a simple, well known strategy). The PQL query itself has not to be changed to enable replication (e.g., no need to insert a `ReplicationMerge` operator manually).

Odysseus Script example for replication.

```
#PARSER PQL

#CONFIG distribute true

#PEER_PARTITION QUERYCLOUD

#PEER_MODIFICATION REPLICATION 2

#PEER_ALLOCATE ROUNDROBIN


#ADDQUERY

proj = PROJECT({

                  attributes=['auction', 'price']

                }, nexmark:bid)

max = AGGREGATE({

                  aggregations=[['max', 'price', 'maxPrice']],

                  group_by=['auction']

                }, proj)

out = SINK({

             Sink='MaxPrice'

                }, max)
```

Besides the replication of a complete query, the user can decide to replicate just a subset of the operators. Listing 2 shows the query definition in PQL. It's the same query as in Listing 1 with the exception that the replication mechanism has an additional argument. That argument defines the start and end point of the replication (the replication will begin after the start and end before the end). Values for start and end can either be a source name (e.g., nexmark :bid; only allowed for start) or a unique ID of an operator.

Odysseus Script example for partial replication.

```
#PARSER PQL

#CONFIG distribute true

#PEER_PARTITION QUERYCLOUD

#PEER_MODIFICATION REPLICATION [start,end] 2

#PEER_ALLOCATE ROUNDROBIN


#ADDQUERY

proj = PROJECT({

                  attributes=['auction', 'price'],

                  ID = 'start'

               }, nexmark:bid)

max = AGGREGATE({

            aggregations=[['max', 'price', 'maxPrice']],

            group_by=['auction']

         }, proj)

out = SINK({

                  Sink='MaxPrice',

                  ID = 'end'

               }, max)
```