TimeInterval

Time interval is the most important meta data in Odysseus as it is the base for a semantically correct processing. It is used for applying temporal context to events like in most streaming and event based systems. Without going into deep detail, it allows to map the semantic of streaming operators to the semantic of relational algebra operators and to apply a so called snapshot reducibility.

The time interval is defined with a start time stamp and an end time stamp, and describes the temporal validity of the event. Events are only allowed to be processed together if their time intervals overlap (they are valid during the same time). We will give an example below.

You can access the meta data from every stream element with `TimeInterval.start` and `TimeInterval.end` (e.g. in MEP or in Select operator)

To set the start time stamp of incoming events, each source in Odysseus must be described (similar to the create table command in SQL), especially a schema giving the attribute names and their data types is needed for tuples. By using the special data type starttimestamp the content of this attribute is interpreted as time stamp and the meta data of the tuple will be set to this application time value. If no information is given, the value is interpreted as milliseconds since 1970, as this is the default case in many systems (e.g., Unix). If no attribute is dened as starttimestamp, the current system time is used as start time stamp.

The start time stamp can be manipulated during the processing. This can be done with a special TIMESTAMP operator or because the semantic of an operation requires the adaption. E.g., the Timestamp operator operator may be used if an input attribute needs some processing before it can be interpreted as time. A typical example is a string based time stamp.

The end time stamp states the point in time when an event gets invalid. Initially, the end time stamp is set to innity. This means, the event starts at some time and is valid forever. To set the end time stamp, dierent options exist in Odysseus. Similar to the starttimestamp an endtimestamp attribute data type can be used. The typical case however is the usage of windows, which reduce the validity of an event to a distinct portion of time.

In opposite to many other systems, the denition of a window in Odysseus is not coupled to the operators that use these windows (e.g., a 15 minute aggregation). Instead, window operators are provided. We allow time based (e.g., the last 30 minutes), element based (e.g., the last 100 events), and predicate based (e.g., only when the temperature is above 10 degree centigrade) windows. For all window types further options like the movement or a grouping are possible. Each window denition is mapped to a modication of the end time stamp. The main advantage of this approach is that following operators do not have to deal with the way a window is dened.

How an operator handles a time interval depends on the operator type. For most stateless operators like lters, mappings or projections, the time stamp is ignored and not manipulated in any way. Instead, statefull operators need to take the content of the time interval into account. There a two ways of treatment. For an operator like Union operator the time intervals are not manipulated. Typically, Union operator is in most systems a stateless operator. In our system, we need some hint of time progress for purging reasons. This can be done with so called punctuations or simply by ordering all events according to their start time stamps. Irrespective of the way, order needs to be assured (at least for the punctuations). So the Union operator needs to keep state about the time progress in each of it input streams and has to guarantee that no events occur out of order.

The Aggregate (and Group) operator is an example for the other way of treatment. It has to assure that only events with overlapping time intervals are aggregated. The result of the aggregation is a new event with a time interval that is the intersection of all involved events.

The Join operator operator is a mixture of both ways of treatment. A JOIN merges two events from dierent input streams if thejoin predicate is fullled. Additionally, it has to consider the time intervals. Only events that are valid at the same time are allowed to be joined and the time interval of the resulting event is the intersection of the time intervals of both events. Since the intersection operation can produce out of order events, the Join operator also has to handle order related tasks as the Union operator does.