Features

This page lists some of the features and components that are provided by Odysseus.See How to install new features for explanation of how to install a new feature.

Notice that this list is neither complete nor may contain some functions that are not working at the moment. We distinguish between core concepts that are generally given by its framework architecture and additional features that use the framework to extend Odysseus with new concepts and functionalities.

The Core Feature: Basic Concepts and Components

The core feature is the basis of Odysseus. It holds all stuff that is needed by Odysseus in any case, e.g. that the query processing needs a transformation process etc. Since Odysseus is a framework, its components can be extended and configured via services and interfaces. However, the core also includes the most common implementations for the components, so that there is at least one complete system configuration. In the following, we list some of the basic framework concepts of Odysseus and say how a framework concept is implemented by a dedicated technology/concept

Component-based Architecture

Odysseus is based on OSGi so that it is a component-based architecture. Most of the functionalities in Odysseus can be extended or configured via services through the components. This also allows the adaption of the system during runtime. Since it is also possible to substitute components, default concepts can be replaced by other/own concepts.

Independent Processing Objects

Most systems only have one processing type (e.g. relational, XML or just strings). Odysseus can handle arbitrary processing types. The default processing type is a "relational".

Extendable Metadata (TimeIntevals and Latency...)

Each object that is processed by the system can be enriched with metadata. Besides fixed metadata, each object can be optionally annotated with key-value pairs. The fixed metadata is not optional, because it is for example used for the processing. Therefore, the default metadata is "TimeInterval" (or also referred to as "interval" or "intervalapproach"). The TimeInterval metadata provides two timestamps that indicate the start and the end of the validity of the processing object. Each operator that recognizes the timeinterval metadata uses this metadata to process only those processing objects that are valid at the same time. This allows the processing of an potentially unbounded data stream by windowing the stream through time intervals. Furthermore, there is the possibility to use a latency metadata - which is used for measuring the latency of a processing object.

Arbitrary Schema and Datatypes

The processing objects can be specified by a schema and datatypes that is extendable and is called simple description framework (SDF). In the relational processing, for example, the schema describes the names (attributes) and the datatypes of the tuple (which is the processing object in the relational world). The schema can be seen as a list of attributes and each attribute has a name and a datatype. Although there are datatypes for integer, float or something else, it is also possible to introduce your own datatype. This could be everything and is only a marker that could be used by operators, but neither the schema nor the datatypes are normally used by the processing (Needless to say, that some relational operators during the relational processing, e.g. a projection, need the schema, so that it have to fit to the processing object).

Transformation: Logical and Physical Operators

Odysseus distinguishes between logical operators that only say "what" this operator does but does not say "how". Thus, the logical layer is independent from any implementation and normally also from any processing object types (see above). The transformation converts the logical into a physical representation. This physical counterpart provides the real implementation. Thus, it is possible to have more than one implementation for one logical representation. A set of rules and a rule engine manages how a logical operator is transformed and which physical operator is used. Since these rules can be complemented and overloaded, it is also possible to announce own rules (e.g. to transform the logical operators into physical operators for a new processing type)

Rewrite: Optimization of Logical Operators

A given logical plan (which is a graph based on logical operators) can be optimized via restructuring using a set of rewrite rules. For the relational processing for example, there is a rule that pushes a selection down to the source, so that the number of processing objects is reduced as early as possible. These rules can also be overloaded and completed by new ones.

Creating New Operators

It is possible to create your own new operators. For this, the only thing is to create a logical and a physical operator and an appropriate transformation rules that says how this logical operator should be transformed into the physical operator.

Build-in Math Expression Parser (MEP)

A math expression parser (MEP) provides the possibility to evaluation expressions like "(a + 5)*93/12". MEP can be extended by new functions (like round, floor, abs...)

Several Scheduling Strategies

Odysseus provides several scheduling techniques which can be extended. The default scheduler is a single thread scheduler who schedules all operators by one thread. however, it is also possible to split up the plan into several thready. There are also several scheduler strategies available like "aurora min latency/memory/cost" or a round robin.

Extendable Parser

The parser is used for transforming a query (normally its a string) into a logical query plan. It is possible to have multiple parser at one.There exists, e.g. PQL and CQL. PQL is a default parser where each operator can be expressed via a procedure. CQL is based on SQL and is similar to StreamSQL. It is also possible to integrate new languages. Furthermore, PQL can be easily extended for new operators by annotating the logical operator.

Webservice and Console Executor

The executor manages all things (installs and runs query or adds and removes sources). So the executor is the interface for external accesses. It can be used via code, a webservice or a console. However, it is also possible to extend the executor to provide a new accessibility. The webservice interface, for example, can be used by other applications (even non-Java) to access to Odysseus.

Odysseus Script

An own script language that is called "Odysseus Script" provides the possibility to run a set of queries or setting parameters through one script-file.

Access Framework

The access framework of Odysseus is responsible for creating source and sink operators. For example, a source operators is used for connecting to a sensor to open a data socket where the sensor can push its data. The access framework provides several layers/parts which can be combined to build a suitable access operator. For example, there is a transport layer that describes how the data is provided (e.g. a tcp socket, as a file or through a serial port). Based on this, a protocol handler tells how the data is represented (e.g. as lines or text or byte buffers). This protocol handler uses a set of data handlers which say how the text or the bytes are transformed into the data. All these handlers can be extended as well.

Buffer Placement

Sometimes it may necessary that operators are not directly coupled. For this, it is possible to insert some buffer operators. Although they can be inserted by hand (e.g. via PQL), a build in buffer placement mechanism can be activated so that buffers are automatically placed into the query plan during the installation. There are also several buffer placement strategies like for each source or for each operator.

Query Sharing

Query sharing is an optimization technique where Odysseus reuses existing partial plans when a new query plan is installed. For this, it only reuses a partial plan, if new and existing operators are semantically equal.

User Management

Odysseus is a multi user system. For this, each installed query or source is dedicated to a user. Since more than one user can access the system, it is also possible to grant or revoke special rights to other users.

Punctuations / Heartbeat Mechanism

Odysseus has a built-in punctuation mechanism (in the relational processing). Since the window concept of the interval approach may cause a blocking of operators, because a processing step of an operator needs further elements to produce results. However, if there are no further elements, the processing blocks. At this point, punctuations indicate that the stream is still "alive" but there are currently no elements (thus, also called heartbeat). So, punctuation are used to unblock the operator earlier.

Additional Features

Admission Control Feature

The admission control pays attention to the system load when new queries are added. For example, if the system load is too high, a new queries could be refused by the system. For this, it estimates the load of a query by measuring the selectivity and data distribution.

Action Feature

The action feature allows to invoke some actions via webservice - e.g. send a command to an actuator

Benchmark Feature

The benchmark feature offers some operators that are able to measure, e.g. the latency for a given query plan. Furthermore, there exists a benchmark runner that can start Odysseus and run some queries several times.

CEP (Complex Event Processing) Feature

The CEP in this terms provides the possibility to recognize patterns. This can be used, for example, if you want to recognize if a value raises up for some time until it falls for 30%. For this, there is a CEP operator available that can be configured via a SASE+ pattern language

Further Information about the SASE.

Context Feature

The context feature provides a context store that can be used to hold current valid status (e.g. the door is currently open). A store and enrich operator is provided for modifying the store via streams.

RCP Feature

The RCP feature provides an user interface for Odysseus and is also called "Studio". Besides an editor for running Odysseus Scripts, it allows to visualize streaming data via different charts, tables or lists. It also gives the possibility to show current status like running queries, installed sources, known users or the visualization of the current query plan. Furthermore, the RCP is also extendable with new views like tables, charts or own views.

Costmodel Feature

The costmodel measures some statistics to estimate the behavior of an incoming stream and the selectivity of operators. This allows one to estimate the cost of query plans. This can be used, for example, in the admission control or also for optimization techniques.

Database Feature

Although Odysseus is designed for streaming data, there could be also some static data that is (usually) stored in a database. Therefore, this feature provides drivers and interfaces for opening connections e.g. to a MySQL, Oracle or PostgreSQL database. Furthermore, there are for the one hand a sink and a source operator to write or read static data as streams from the database. For the other hand, there is an enrich operator to enrich streaming objects with static data from a database.

Priority Feature

This feature allows the priorization of processing object, e.g. when there are alerts. If this feature is enabled and there are elements with a higher priority than other (normal) ones, they can overtake the normal elements to be processed faster.

KeyValue Feature

The KeyValue Feature allows to read, process and write data as key-value pairs. It includes also wrappers for JSON and BSON data handling. (more information)

Mining / Learning Feature

This feature provides concepts for learning from data streams. Besides clustering and classification, there exists also a concept for learning frequent patterns from the stream.

Non-distributed Recovery Feature

This feature bundles all available recovery techniques for thenon-distributed version of Odysseus. Among other things it contains techniques for a gap recovery (at-most once), a rollback recovery (at-least-once) and a precise recovery (exactly once).

OdysseusNet Feature

This features encapsulates concepts to use Odysseus in a distributed network containing multiple machines.

P2P Feature

This feature distributes Odysseus over a network of peers

Probabilistic Feature

This feature allows the processing of discrete and continuous probabilistic values.

Prototyping Feature

With this feature, it is possible to write "user defined aggregates" and "user defined functions" in several script languages like Java-Script or Ruby.

Scheduling Feature

As mentioned above, the scheduling feature installs several scheduling mechanism like a priority-based or an aurora-based scheduling

Security Punctuation Feature

Security punctuations are points in a data streams that tells the processing who is allowed to read the data. Thus, the stream itself can define rights like "as from now you are not allowed to see the next 100 tuples".

SLA Feature

This feature allows the definition of service level agreements, which is for example used for a SLA based scheduling

SPARQL Feature

Streaming SPARQL is an extension to SPARQL for querying on RDF data streams

Spatial Feature

The spatial feature introduces spatial data types like points, lines, polygons, etc., and defines some functions on them, e.g. covers, crosses etc. Thus, a geographic processing is possible.

Test Feature

The testing feature runs Odysseus, starts some predefined queries, inputs some data and checks, if the output is as expected. Thus, this can be seen like unit/black box testing

Interval Feature

The Interval Feature adds data types and functions to work with intervals.

Wrapper Feature

As mentioned above, Odysseus has an access operator framework that allows to build different types of adapters. This feature provides some predefined adapters, for example for CSV, Files, HTML, JSON, NMEA, Google Protobuf, RS232 (serial port), TCP socket or Twitter. Furthermore, there is a set of handlers for data handling like for integers or strings etc. There are also concepts for a push and a pull based processing.

XML Feature

The XML Feature contains different operators for processing XML documents.