The Odysseus Rule Engine

The Rule-Engine provides basic concepts for a rule-based processing for other compoents. For instance, it is used during rewriting and transformation of queries.

Components and semantics

The Rule-Engine has several components

Working Memory

The working memory exists of all objects that have to be processed by rules. For the transformation, for example, the working memory consists of all logical operators that should be transformed by rules into their physical counterpart.

To interact with the working memory in a rule, there are the following methods:

- **insert**: adds a new object to the working memory.
- **retract**: removes an object from the working memory
- **update**: informs the working that an object has been changed
- **updateAll**: runs update for a collection of objects
- **getCollection**: Returns a list of all objects that are currently in the working memory
- **getAllOfSameTyp(K k)**: This method returns all objects that have the same type like the given object k. For example, *getAllOfSameTyp(new SelectAO())* can be used to return all SelectAOs that are currently part of the working memory

Important is the retract method. If an object is never removed from the working memory by using retract, although there are rules, it may be possible to get into an infinity loop!

Inventory

The inventory consits of all necessary objects for the current rule engine instance. This is, normally, the set of all rules and the RuleFlowGroups (see below).

Each time when the rule engine is invoked, the rule engine creates a copy of the inventory. This is needed so that rules can not be changed if they are in use. Furthermore, it allows nested runs of the rule engine, e.g. to run a transformation within another transformation without influecing each other.

The rule engine consists of a two-stage hierarchy: First, there is a list of RuleFlowGroups and, secondly, each RuleFlowGroup has a set of Rules.

RuleFlowGroup

The rule enginge allows to create groups of rules, which are named RuleFlowGroups. Each Rule has to be assigned to such a group. The rule engine runs one group after another group. Therefore, each instance of the rule engine (e.g. the one for the transformation) has a fixed ordered list for this groups . However, a rule group may occur more than one in this ordered list.

If the rule engine is executed, it runs each group in succession. For each rule flow group, each rule of this group is tested if it is executable. Therefore, there are two possibilities:

- If none of the rules of the current rule flow group was executed, the rule flow group is finished and the next rule flow group is started
- If at least one of the rules of the current group was executed, the current run of the grouped is discontinued and the whole group is started from the beginning. This means, rules that were not executable before are tested again.

Rules

Each rules belongs to a RuleFlowGroup. Furthermore, each rule is responsible for a dedicated type, which is defined by a generic within the rule (this might be, for example, a certain operator).

For each run of a RuleFlowGroup, the following steps are performed for each rule of the group:

- It is checked, if one of the objects in the working memory fits to the dedicated type of the rule
    - If it does not match the type: check the next object in the working memory
    - If it matches the type: run the isExecutable method
        - if isExecutable returns false, break and run the rule for the next object in the working memory
        - if isExecutable returns true, run execute and (like it is written above) the RuleFlowGroup is restarted.