Constraint-Aware Plan Implementation Details

Notes

- Since the composition plans were pruned in the previous stage, some service node objects may contain predecessor and successor lists with pointers to nodes that do not exist in their container plan. Such pointers are removed from these lists while transforming a simple composition plan into a constraint-aware composition plan.
- Due to pruning, some composition plans may contain empty service layers. While transforming a simple composition plan into a constraint-aware composition plan, empty layers are removed from the plan and the layer indexes of all the service nodes in the plan are adjusted according to the new layer sequence.
- A constraint object is assigned to a service node only once. Point to be noted here is that distinction between constraints is made based on their Java objects and not on their elements/features/data members.

Assumptions

- All the constraints have been assumed to be unique. No technique for differentiating between constraints (other than eliminating duplicate Java objects) has been used in the present solution. Even same constraints, if defined as different objects, will be executed explicitly.

Future Work

- More complex solutions can be employed to differentiate between constraints and eliminate the duplicates from a service node. This would reduce the amount of constraint evaluation, thus reducing the execution time of a constraint-aware composition plan.
  A simple comparison technique could have been to compare the type, operator and literal values of constraints respectively. However, this does not necessarily detect duplicates. For example, by this technique, constraint "price < 100" is different from "price <= 99" whereas mathematically both the constraints denote the same range of values for "price". Due to such limitations, no simple technique for differentiating between constraints (other than eliminating duplicate Java objects) has been used in the present solution.