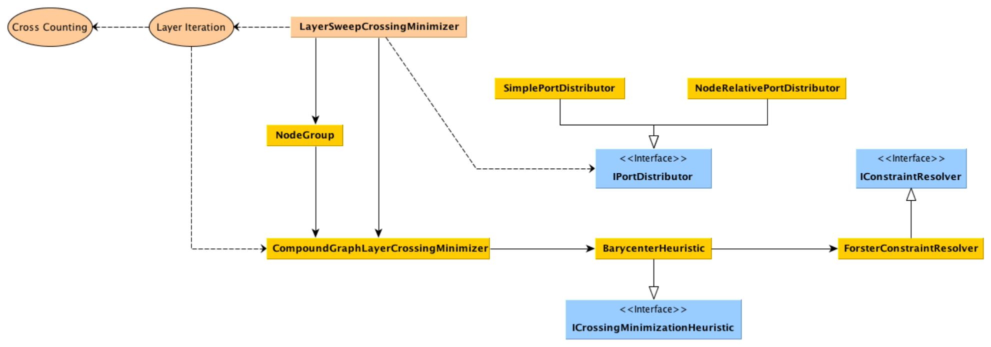
Layer Sweep Crossing Minimization

Crossing minimization is hard to get right and hard to understand once written. Case in point is our layer sweep crossing minimizer, that implements layer sweep crossing minimization in a monolithic bunch of some 1,200 lines of gloriously hard-to-understand code. (at the time of writing) A new structure is required!



The new structure factors out much of what can possibly be factored out:

- `LayerSweepCrossingMinimizer` is still the main class and implements the layer-sweep approach to crossing minimization. However, it does not know anymore how to do the actual crossing minimization step for a given pair of layers. However, it implements the cross counting algorithms, since these are not affected by the actual crossing minimization algorithm used.
- An `ICrossingMinimizationHeuristic` is used to determine the ordering of a given set of nodes. The result is expected to adhere to any constraints, such as node successor constraints and layout unit constraints. The usual heuristic would be the `BarycenterHeuristic`.
- The `IPortDistributor` is used by the `LayerSweepCrossingMinimizer` to calculate port ranks needed to determine node barycenters, and to handle the final port distribution. The standard implementation is the `NodeRelativePortDistributor`. We're planning to add a `SimplePortDistributer` as well that calculates port ranks as KLoDD did back in the day.
- If a heuristic doesn't know how to handle constraints, it can make use of an `IConstraintResolver` to resolve any constraints after it has determined an initial node ordering. The standard implementation is the `ForsterConstraintResolver`, which implements a constraint resolving heuristic proposed by Michael Forster.
- `NodeGroups` are used by the `CompoundGraphLayerCrossingMinimizer` to encapsulate sets of nodes contained in a single compound node, to treat them as an atomic group with fixed ordering.