Introduction to the Dependency Graph (Version 2)

By: Saba Alimadadi Jani

If we look at the dependency graph structure from a high-level point of view, we will see nodes and edges. Nodes contain the content (the data) and edges represent the dependency relations between the nodes.

Below, the important classes inside the code are briefly described, along with their most important fields and functionalities.

Node

Nodes generally contain the data in the graph. A node can have an operation, input node(s), and output node(s). The output nodes are the nodes that depend on this node. The input nodes on node n, however, are the nodes that n depends upon. A node can have none/some/all of these fields set. They can be set when the node is instantiated, or can be added later on in the analysis process.

Operation

Operation is a main field of each node. Each operation can have OperationInfo, input and output ports. Operation info contains some information about the current operation and can be used (or extended) by the user. Ports are value containers; input ports are the ports that the operation needs in order to perform. As a result of the operation being performed, it can affect the output port. In both cases, operation can use any subset of the input/output ports.

Each operation that the user implements must extend the dependency graph's Operation class. Therefore the user is obligated to implement the evaluate() method. In this method, the user defines what this special type of operation actually does, setting the output port(s) using input port(s).

Port

This class mainly contains a value. The value is generic so the value itself and the appropriate operations could be set and defined based on user's needs. The ports are used as input/output for the operations.

Dependency Graph

The dependency graph keeps a list of all nodes. It also knows all the listeners that are waiting for an event happening in the dependency graph.

The dependency graph handles nodes and the dependency relations between them. It has an update method that propagates the change through the whole graph based on an updated node; resulting in all relevant values to be updated. The update process first performs a topological sort for all the nodes in the graph based on the updated node. It then fires the update for each node that depends on the

updated node, directly or indirectly, based on the sorted list. And the update happens based on ports being modified (not based on nodes alone).

Events

Every meaningful action that happens in the dependency graph (such as adding a node, adding an edge, or update) fires an appropriate event, to be used by another party that wants to use the dependency graph, e.g. a visualization, another software, etc. in the events package, there is a base class for all dependency graph costume events, and all dep graph events extend this base class. There is also a listener class for dependency graph events that catches them and performs an appropriate action. The users can change the action performed for each event based on their needs.