Copyright © Sharada Prasanna Mohanty, 2016
All Rights Reserved

Chapter 1

Kathaa Orchestrator

1.1 Kathaa Orchestrator

Kathaa Orchestrator is at the core of the whole Visual Programming Framework. Kathaa Orchestrator obtains the structure of the Kathaa Graph and the initial state of the execution initiator modules from the Kathaa Interface, and then it goes on to efficiently orchestrate the execution of the graph depending on the nature and state of the modules, while dealing with process parallelisms, module dependencies, etc under the hood.

1.2 Core components

1.2.1 Job Queue

The Kathaa Orchestrator maintains an internal job queue which it executes on the resources available to it. All the jobs are expected to be parallel, and in principle can have shared resources. The rest of this chapter refers to the Job Queue using the variable `JOBS`.

1.2.2 Core functions

1.2.2.1 incoming_edges(G, M)

This is a function which for a given Kathaa Graph `G` returns a set containing all the incoming edges into a particular Kathaa Module `M`.

1.2.2.2 outgoing_edges(G, M)

This is a function which for a given Kathaa Graph `G` returns a set containing all the outgoing edges into a particular Kathaa Module `M`.

1.2.2.3 non_optional_edges(G, M)

This is a function which for a given Kathaa Graph `G` returns a set containing all the non optional ports (as defined in the meta structure of the module) for a particular Kathaa Module `M`.

1.2.2.4 sentence_input_nodes(G)

This is a function which for a given Kathaa Graph `G` returns a set containing all the Module Instances or Nodes in the graph of the type `sentence_input`.

1.2.2.5 resource_nodes(G)

This is a function which for a given Kathaa Graph `G` returns a set containing all the Module Instances or Nodes in the graph of the type `kathaa-resource`.

1.2.2.6 CHECK_DEPENDENCY(G,M)

This function(Algorithm 1) checks if all the dependencies of a particular Module Instance or Node are satisfied.The pseudo code for this function can be summarized as follows :

Algorithm 1 Function to check if the dependencies of a particular Module Instance or Node in a Kathaa Graph are satisfied

```
1: procedure CHECK_DEPENDENCY(G, M)
2:     dependency_satisfied ← True
3:     for e in incoming_edges(G, M) + non_optional_edges(G, M) do
4:         input_port ← e.source_port()
5:         if input_port.value() == Null then
6:             dependency_satisfied ← False
7:     returndependency_satisfied
```

1.2.2.7 UPDATE_INPUT_PORT(G, M, input_port, Value)

The function(Algorithm 2) updates the Input Port values for Kathaa Module instances at different points of the execution of a Kathaa Graph. The pseudo code for this function can be summarized as follows :

1.2.2.8 PROCESS(JOB)

A background function listens to events in the Job Queue `JOBS` and uses this function(Algorithm 3) to process the Jobs as soon as they are queued (and also based on the availability of resources).

Algorithm 2 Function to update input port values for all Kathaa Module instances

```
1: procedure UPDATE_INPUT_PORT(G, M, IP, Value)
2:     input_port.value ← Value
3:     if CHECK_DEPENDENCY(G, M) == True then
4:         JOB = new JOB()
5:         JOB.set_graph(G)
6:         JOB.set_node(M)
7:         JOBS.queue(M)
```

Algorithm 3 Function to process Jobs from the Job Queue

```
1: procedure PROCESS(JOB)
2:     M ← JOB.node()                                          ▷ Module Instance
3:     G ← JOB.graph()
4:     try
5:         M._function() ▷ Run the actual computational function associated with the Node
6:         for e in outgoing_edges(G, M) do
7:             source_node ← e.source_node()
8:             source_port ← e.source_port()                   ▷ Output port at the Source Node
9:             target_node ← e.target_node()
10:            target_port ← e.target_port()                   ▷ Input port at the Target Node
11:
12:            UPDATE_INPUT_PORT(G, target_node, target_port, source_port.value())
    ▷ Copy the data from the output port at the source node to the input port at the target node
13:        JOB_COMPLETE(G, M, success, result)
14:        return (False, M)  ▷ return a tuple representing a boolean state for the successful execution of the function, and the updated Module Instance M
15:    catch Exception E
16:        JOB_COMPLETE(G, M, success, result)
17:        return (True, E)    ▷ return a tuple representing a boolean state for the successful execution of the function, and the error object E
18:    end try
```

1.2.2.9 JOB_COMPLETE(G, M, success, result)

This function(Algorithm 4) is an event handler to deal with the post processing steps involved when a Job is completed.

Algorithm 4 Event Handler for Job Complete

```
1: procedure JOB_COMPLETE(G, M, success, result)
2:     Visual_Interface = G.get_visual_interface()
3:     Visual_Interface.update(G, M, success, result)
4:     JOBS.remove_node(M)
```

1.2.2.10 EXECUTE_GRAPH(G, M)

This function(Algorithm 5) starts the execution of a Kathaa Graph `G`.

Algorithm 5 Function to start execution of a Kathaa Graph

```
1: procedure EXECUTE_GRAPH(G, M)
2:     if M not NULL then          ▷ When a Node to start the execution from is provided
3:         JOB = new JOB()
4:         JOB.set_graph(G)
5:         JOB.set_node(M)
6:         JOBS.queue(JOB)
7:     else      ▷ When a Node to start the execution from is not provided, start with all the
   sentence_input nodes and the resource nodes
8:         for m in sentence_input_nodes(G) + resource_nodes(G) do
9:             JOB = new JOB()
10:            JOB.set_graph(G)
11:            JOB.set_node(m)
12:            JOBS.queue(JOB)
```

1.3 Quick Summary

As a quick summary, the execution of a Kathaa Graph starts by collecting all the Nodes in the Kathaa Graph from which it should start the execution. In the case that the Visual Interface provides a Module Instance to begin execution from, the Kathaa Graph starts execution from just that Node, else it collects all the `sentence_input` nodes and the `resource` nodes. The collected nodes are simply queued in a global Job Queue. A background process, in the meantime, listens on the Job Queue, and whenever a Job is added to the queue, it tries to execute the Job in any of the available resources. The execution of the Job starts by trying to execute the actual function associated with the particular Node, and if its successful, it passes the data along all its outgoing edges to the designated input ports of module instances further

along the graph, and then finally returns the obtained result object along with the success state set as `True`. If there were any errors in the execution of the supplied function, it returns with the success state of the function as `False` and the actual exception that occurred during the execution of the associated function. In both the cases, the Job Complete event handler is called with the success state and the final result (or the exception in case of any errors), and the Job Complete event handler passes along the data to the Visual Interface to update the state of the graph in the Visual Interface and provide the user with the result associated with the Module or the actual exception and the error message to help the user debug the particular Kathaa Graph.