**Assignment-1 Report**

**Course:** ​ Advanced Operating Systems (CS\_G623)

**Title:** ​Huang’s Termination Detection Algorithm

**Group No.:** ​3

Akalankam Sai Teja - 2020H1030142P

Shahid Nazir - 2020H1030120P

Atul - 2020H1030138P

**Table of Contents**

[1. Abstract 1](#_Toc60138592)

[2. Introduction 1](#_Toc60138593)

[3. Description of Algorithm 2](#_Toc60138594)

[3.1 Assumptions of the algorithm: 2](#_Toc60138595)

[3.2 Notations used in the algorithm: 2](#_Toc60138596)

[4. Example 3](#_Toc60138597)

[5. Simulation Implementation 6](#_Toc60138598)

[5.1 Network Construction 6](#_Toc60138599)

[5.2 Flow of the program 6](#_Toc60138600)

[6. Diagrams 10](#_Toc60138601)

[6.1 Class Diagram 10](#_Toc60138602)

[6.2 State Chart Diagram 11](#_Toc60138603)

[6.3 Sequence Diagram 12](#_Toc60138604)

[6.4 Collaboration Diagram 13](#_Toc60138605)

[7. Test case specification 14](#_Toc60138606)

[7.1 Objectives: 14](#_Toc60138607)

[7.2 Preconditions: 14](#_Toc60138608)

[7.3 Input specification: 14](#_Toc60138609)

[7.4 Output specification: 14](#_Toc60138610)

[7.5 Post conditions: 15](#_Toc60138611)

[7.6 Conclusion: 15](#_Toc60138612)

[8. Advantages and Disadvantages of Huang’s Algorithm 15](#_Toc60138613)

[8.1 Advantages 15](#_Toc60138614)

[8.2 Limitations 15](#_Toc60138615)

# Abstract

Detecting termination of a distributed computation is a fundamental problem in distributed systems. This report presents the implementation of Huang’s algorithm for detecting computation termination in a distributed system. The algorithm assigns a weight to each active process and to each message in transit. The controlling agent has a weight, too. The algorithm maintains an invariant that the sum of all the weights related to a computation is equal to one. The Controlling agent, when it receives all the sent messages back from participating processes, declares that a computation is over.

# **Introductio**n

A distributed computation consists of a set of processes, which communicate with one another via message passing. Each process may be either active or idle. An active process may become idle at any time. Only active processes may send messages to others; and, an idle process can only be reactivated by receiving a message. The computation is said to be terminated if and only if all the processes are idle and there is no message in transit.

This report presents an implementation of the Huang’s termination detection algorithm. The algorithm implementation has been done in java language using Daj framework.

# Description of Algorithm

The assumptions that the algorithm makes are given below.

## ****3.1 Assumptions of the algorithm:****

* One of the co-operating processes which monitors the computation is called the controlling agent.
* The initial weight of controlling agent is 1
* All other processes are initially idle and have weight 0.
* The computation starts when the controlling agent send a computation message to one of the processes.
* The process become active on receiving a computation message.
* Computation message can be sent only by controlling agent or an active process.
* Control message is sent to controlling agent by an active process when they are becoming idle.
* The sending of a message and the receipt of a message occur as atomic actions.
* The algorithm assigns a weight W (such that 0 < W < 1) to every active process and every in-transit message.

## 3.2 Notations used in the algorithm:

* B (DW): Computation message sent by Controlling Agent to participating process with weight DW.
* C (DW): Computation message sent back by participating process to Controlling Agent with weight DW.

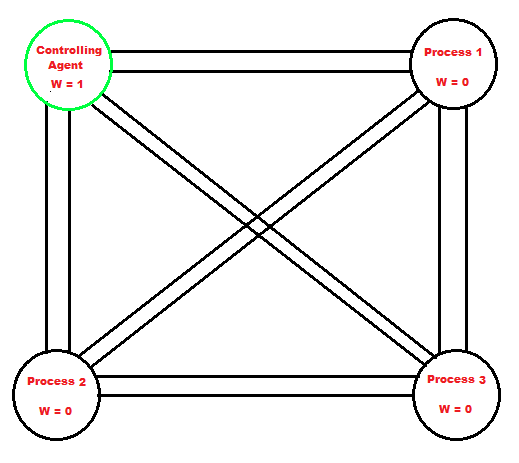
**3.3 Algorithm**

The algorithm consists of the following rules. In the rules, we assume that a weight is infinitely divisible.

* **Rule to send B(DW)**
  + Suppose Process P or Controlling agent with weight W is sending B(DW) to process Q
  + Split the weight of the process P into W1 and W2, such that W = W1 + W2 and W1 > 0, W2 > 0
  + Set weight of the process P as W1 (i.e., W = W1)
  + Send B(W2) to *process Q*, here DW = W2.
* **Rule on receiving B(DW) by process Q**
  + Add the weight DW to the weight of process Q i.e., for process Q, W = W + DW
  + If process Q was idle, it will become active on receiving B(DW).
* **Rule to send C(DW)**
  + Any active process having weight W can become idle by sending C(W) to controlling agent
  + Send a control message C(W) to the controlling agent. Here DW = W.
  + Set weight of the process as 0 i.e., W = 0. (After this process will become idle.)
* **Rule on receiving C(DW) by controlling agent**
  + Add the weight received through control message to the weight of controlling agent i.e., W = W + DW
  + After adding, if the weight of controlling agent becomes 1 then it can be concluded that the computation has terminated.

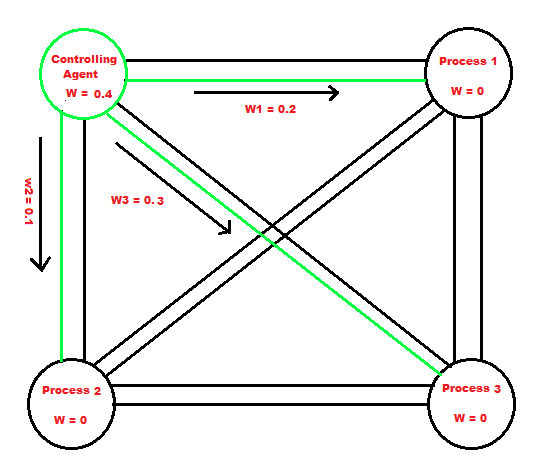
# Example

The example presented in this section illustrates the working of Huang termination detection algorithm.



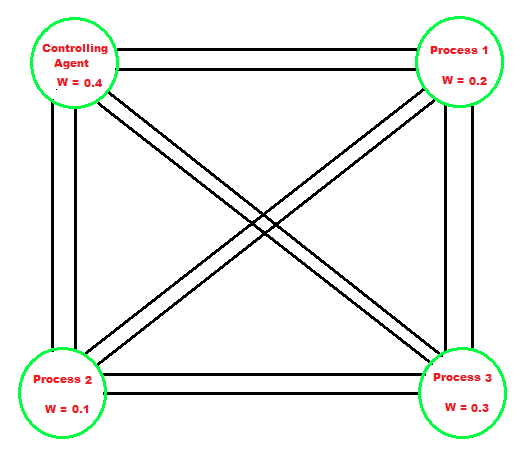
**Figure 4.1**

Figure 4.1 shows a distributed system consisting of four processes. One process will be monitoring the computation and is called the controlling agent and initially only the controlling agent will be active. The initial weight of controlling agent is 1 and all other processes are initially idle and have weight 0.



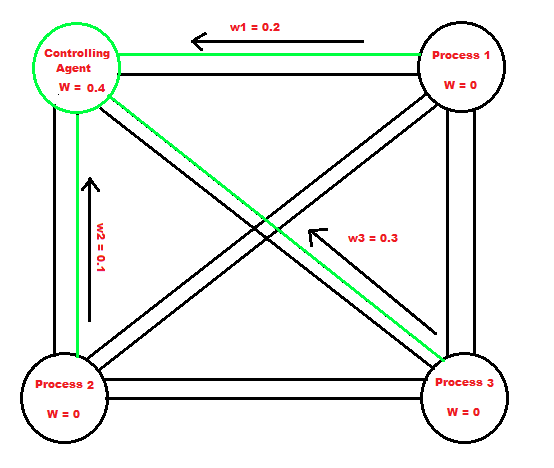
**Figure 4.2**

The Controlling agent selects three random weights w1, w2, w3 (0.2, 0.1, 0.3 in our example) and sends computation messages to P1, P2 and P3. The computation messages will contain these weights and the computation task. This step is shown by Figure 4.2.



**Figure 4.3**

The processes P1, P2 and P3 on receiving these computation messages become active and add the weight from computation message to their own weight. This step is illustrated by Figure 4.3.

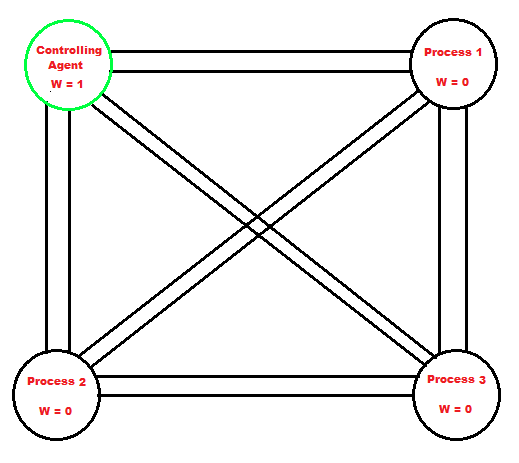


**Figure 4.4**

After the processes have finished the assigned tasks, they will send control messages to controlling agent and set their weight as zero and become idle. The control messages will contain weights. This step is illustrated by Figure 4.4.

After receiving the control message, the controlling agent will add the weight from control message to his weight.

After adding the weight, if the weight of controlling agent becomes 1 then it can be concluded that the computation has terminated. This step is illustrated by Figure 4.5.



**Figure 4.5**

# Simulation Implementation

To simulate the Huang’s termination detection Algorithm, we have used java language. We have also used DAJ toolkit which is a tool for designing, implementing, testing, simulating, and visualizing distributed algorithms.

The simulation implementation has been done using four java classes which are as follows:

* “HTAMain”: This is the main class that has the java main function in it and it is also responsible for creating the network and running the simulation.
* “HTAProgram”: This class contains the implementation of Huang’s termination detection algorithm and object of this class will be running on each process node.
* “HTAMessage”: This class is responsible for creating the computation or control message that will travel in the channels between process nodes.
* “RandomArray”: This class is used to create an array of integers from 0 to 3 in random order.

## 5.1 Network Construction

The network that has been implemented consists of five nodes. Among these nodes one process P0 is the controlling agent and the rest of processes P1, P2, P3 and P4 are normal process nodes. A pair of channels has been created between every process node including controlling agent node. In each channel pair, one channel is used for sending computation messages and second one is used for sending and receiving control messages. This creation of network is done in the main class “HTAMain”.

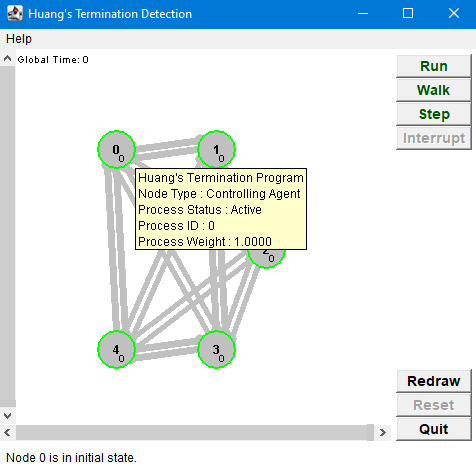
## 5.2 Flow of the program

Each node will run a program which contains the Huang detection algorithm.

The status of each node whether it is active or idle has been done by using a Boolean variable “processIsActive”. If the node is active then this variable will be true and for a idle process this will be false.

Also, one variable “processWeight” will hold the current weight of the process.

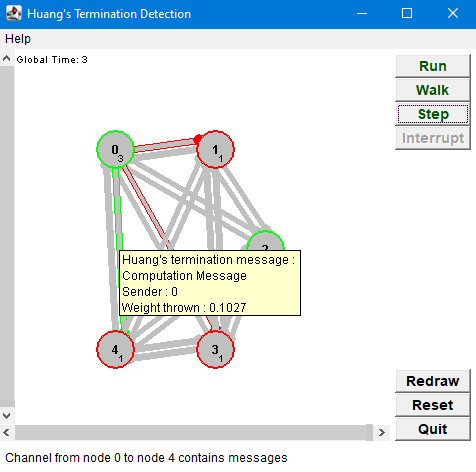
For controlling agent this will be set as 1 and for other process it will be set as 0.



**Figure 5.1 illustrates the initial state of network.**

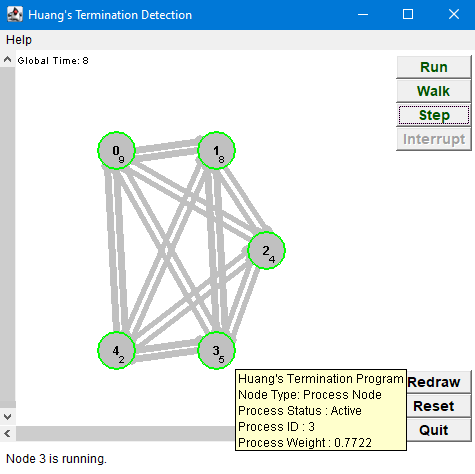
The simulation will be started by the controlling agent which will randomly pick

processes one by one and send computation messages to them. In the message the weight is also sent which is also randomly selected by the controlling agent. The random weight selected is between 0 and current weight of controlling agent. The selection of random processes by controlling agent is done by calling the “generateRandomArray” function from RandomArray class. This function will return an array of integers from 0 to 3 in random order and controlling agent will use this random order to send computation messages randomly. Before sending the computation messages, the weight of controlling agent is reduced.



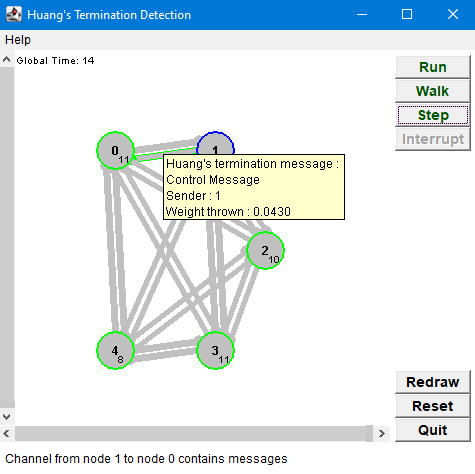
**Figure 5.2 illustrates the sending of messages by Controlling agent.**

The messages which were sent by controlling agent will be received by the process nodes. The process nodes will check the weight in the message and add that to current process weight variable. The process nodes will also change their “processIsActive” variable to true from false which will indicate that the process is now active. To simulate the computation being done by the process nodes, we have used “sleep” function to put some interval between receiving the computation message and sending the control message back to controlling agent.



**Figure 5.3 illustrates the messages have been received by process nodes and computation is being done in process nodes.**

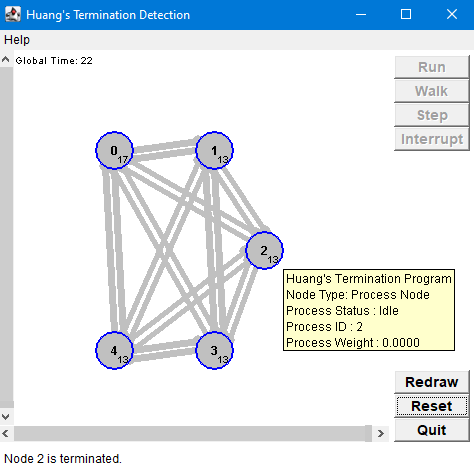
After some time, the process nodes will send the control message to controlling agent which will contain the weight and the process sets its weight to 0. The process also sets its “processIsActive” variable to false which means the process is idle now.



**Figure 5.4 illustrates sending of control messages to controlling agent**

The controlling agent is waiting and will run the program until his process Weight becomes 1 again. He receives the weights one by one from control messages.

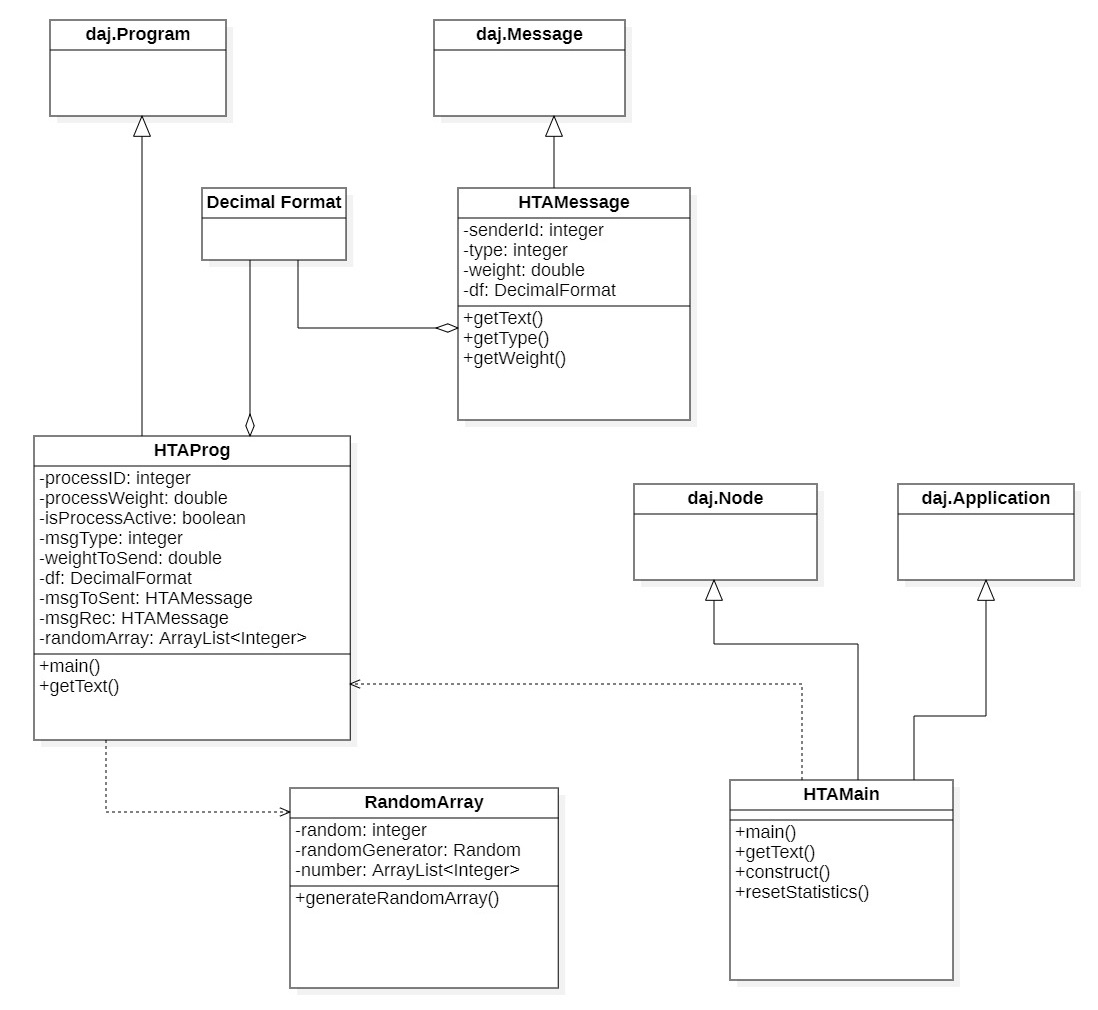
Once the weight becomes 1 again the simulation stops.



**Figure 5.5 illustrates that the termination is detected by controlling agent.**

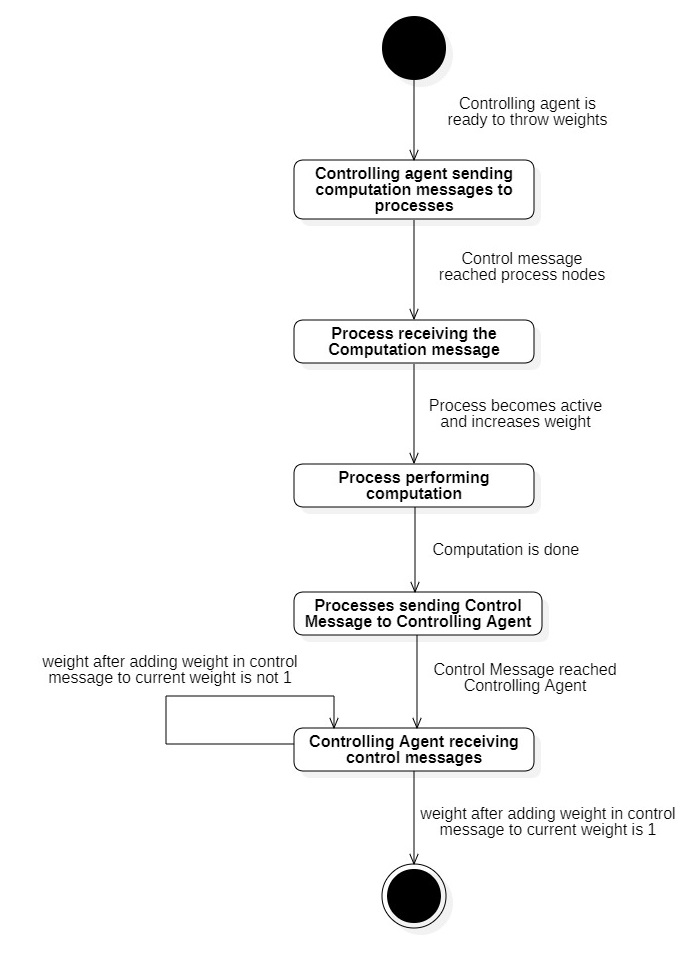
# **Diagrams**

## 6.1 Class Diagram

****

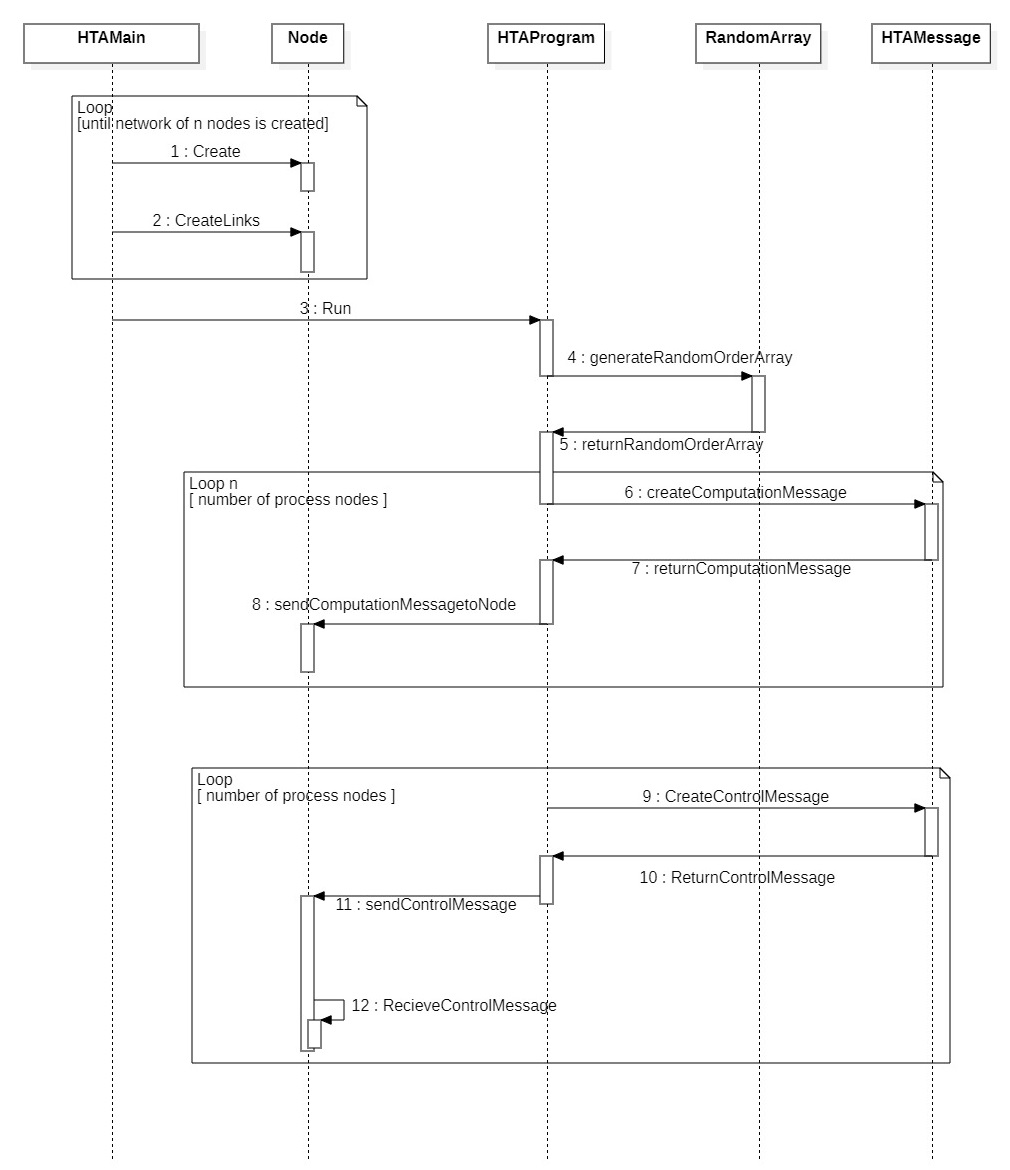
**Figure 6.1:** Class Diagram

## 6.2 **State Chart Diagram**

****

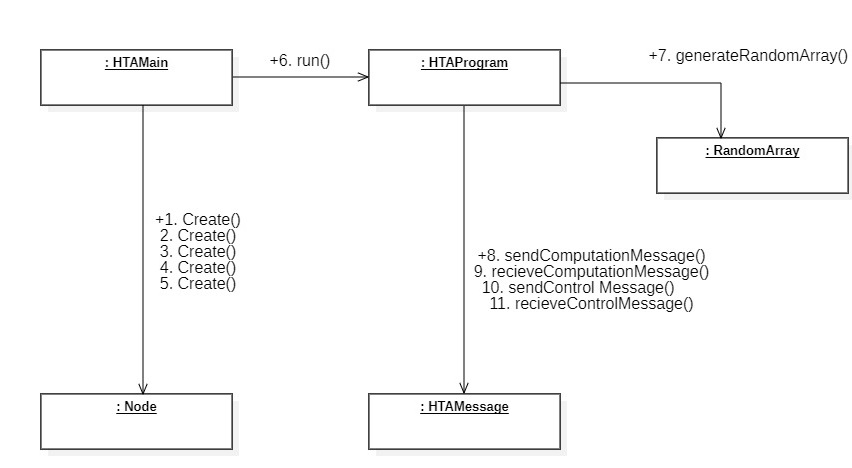
**Figure 6.2:** State Chart Diagram

## 6.3 **Sequence Diagram**



**Figure 6.3:** Sequence Diagram

## 6.4 **Collaboration Diagram**



**Figure 6.4:** Collaboration Diagram

# 7. Test case specification

## 7.1 Objectives:

The purpose of testing this software is identifying the errors, gaps or missing requirements versus the actual requirement.

In our testing process, we would identify:

* + - Whether controlling agent is able to distribute/send the weights to each process node or not.
    - Whether each process node is getting the weights assigned to it.
    - Whether each process node is finishing their computation and returning the weights assigned in confined time or not.
    - Finally, initial weight at controlling node should be equal to the final weight i.e., 1 and the algorithm/computation should terminate.

## 7.2 Preconditions:

The items and documents that were required before executing a particular test case is mentioned in this section.

* + - There shouldn’t be any errors in the code.
    - User should be familiar with how to run the software.
    - IDE which supports java execution.

## 7.3 Input specification:

The input to be given is as follows:

* + - Execute the code(if running in IDE).
    - Click on “Run” to start the execution.
    - Click on “Step” to view the execution stepwise.
    - Press “Reset” to initialize the visualization again.
    - “Walk” lets the program run in slow mode.
    - “Interrupt” suspends the program.
    - “Redraw” would redraw the visualization.
    - “Quit” will terminate the visualization.

## 7.4 Output specification:

The output obtained has to be verified based on these points:

* + - We would verify at each step whether the weight distributed by the controlling agent is getting delivered to the process node or not by hovering over each node to check its attributes and also by hovering over the link to check the weight in transit.
    - Above step should be done for each process node.
    - After the weights have been distributed to each process node, each process node would start to return the weights back to the controlling node.
    - We should check the returning weights in the same way we did while distributing weights.
    - Once all the process node have returned the weight, final weight should be 1 at the controlling agent. This is the final thing that is to be verified.

## 7.5 Post conditions:

* + - Software: If the software is to be run on the browser then java should be enabled.

## 7.6 Conclusion:

Desired results were obtained for the testing process hence we can say that software is working as expected and fulfilling all its requirements and objectives.

# 8. Advantages and Disadvantages of Huang’s Algorithm

## 8.1 Advantages

* The algorithm detects every true termination in finite time.
* It takes only one control message delay for the detection of the process termination.

## 8.2 Limitations

* The algorithm is unable to detect computation termination if a message is lost in transit.
* It also does not work when the controlling agent fails or a process fails while in an active state.