Multi-Agent System Simulator

CISC475/675 Fall 2014 Team 6
Boch Mike
Gotthold Benjamin
Noyes Steven
Sharma Varun
Zimmerman Stefan

Preliminaries

Change Log

- 10/03/1014 - Sprint 2: Implementation Details v0.2
- 09/19/2014 - Sprint 1: Initial Document Release v0.1

Team Information

This project, including the content of this document, have been designed by University of Delaware students Boch Mike ,Ben Gotthold, Noyes Steven, Sharma Varun, and Zimmerman Stefan. To view the current release of this project please visit our project page at http://cisc475-6.cis.udel.edu.

Table of Contents

List of Figures

Chapter 1

Introduction

As technology advances, software is fast becoming the most economical way to test complex solutions to real world problems. Software testing allows a user to run unlimited iterations with complete control over environmental variables. Our Multi-Agent System Simulator (MASS) will provide an environment for Dr. Decker and other researchers to test software Agents allowing them to design more accurate solutions to real world problems faster than ever before.

1.1 Document Purpose

The purpose of this Software Requirements Specification (SRS) is to serve as a statement of understanding between the users of the proposed product and the software developers of the product. The Software Requirements Specification is defined using a subset of the Unified Modeling Language(UML), an Entity-Relationship Diagram(ERD) describing all of the objects/entities along with their attributes, relations, and Data Flow Diagram (DFD).

1.2 Scope

We will develop a general-purpose command line program called ?Multi-Agent System Simulator? (MASS). The MASS will take a Simulation File Input (SFI) and a Configuration File Input (CFI) to produce an environment in which user defined software Agents can connect and interact to solve problems. Upon completion the MASS will output the results of the problem Simulation as well as produce a Log File Output (LFO) detailing the events that occurred within the Simulation. The SFI will be a CTAEMS file describing the task domain that the MASS will use to construct the Simulation environment. The CFI will be a plain text file that contains settings relating to the execution of the MASS but are independent of a given SFI. Agents, which are independent user-defined programs, will connect to the MASS through TCP socket connections before a simulation begins. While running, the MASS allows inter-Agent communication and logs statistics such as the number of messages sent by each Agent. Upon completion the MASS will output the results of the simulation in terms of Quality, Cost, and Duration.

It is important to note that the MASS does not restrict the design of the Agents themselves but requires them to adhere to an connection interface so that communication is possible.

1.3 Acronyms and Abbreviations

This section contains definitions of acronyms and abbreviations used in this document.

- **MASS**: Multi-Agent System Simulator
- **SFI**: Simulation File Input
- **CFI**: Configuration File Input
- **LFO**: Log File Output
- **EE:** Event Engine
- **UML**: Unified Modeling Language
- **DFD**: Data Flow Diagram
- **TCP:** Transmission Control Protocol
- **JSON**: JavaScript Object Notation
- **SRS**: Software Requirements Specification
- **ERD**: Entity-Relationship Diagram

Chapter 2

General Description

The product is meant to serve as a common platform for academic and research oriented activities in the area of Multi-Agent Simulation. The following sections describe the high level view of the system and establish its context. These sections do not state specific requirements but make the specific requirements easier to understand.

2.1 Stakeholders

The stakeholders of the Multi-Agent System Simulator are classified into the following categories.

- **Dr. Decker**: Needs a platform to test his software agents in a way that can help advance his research and lead to new innovations.
- **Dr. Siegel**: Needs a challenging project for his 475 students that is able to be completed over the course of the semester. The project should condone the use of software tools to test implementations and track changes.
- **Researchers**: Researchers might want to use this software for their research or design agents and simulations when testing new ideas.
- **CIS faculty**: Needs an interactive tool for students to make the task of learning more interesting.
- **Developer**: Developers are responsible for the designing, testing, configuring, and upgrading of subsystem.
- **Support**: Support personnel are responsible for the maintenance of the system, software, as well as the installation of subsystems and configuration changes.

2.2 Definition of concepts

- **Multi-Agent System**: A project that includes two or more independent software Agents working to complete a common goal.

- **Multi-Agent System Simulator (MASS)**: The specific computer program being outlined in this document.
- **Simulation**: A generic instance in which the MASS is running on a given input.
- **Agent**: A user defined software program that can communicate with the Simulation through a TCP socket connection.
- **CTAEMS**: A derivative of the TAEMS language that will be employed for specifying task domains.
- **Task**: A high level goal within the system.
- **Subtask**: A low level goal required to complete a Task.
- **Method**: An action taken in order to complete a Task or Subtask.
- **Event**: Individual Tasks, Subtasks, and Methods or a combination of them.
- **Task Group**: A high level grouping of Tasks that share a similar structure or goal.
- **Quality**: A numeric value used to measure the degree of satisfaction resulting from a particular Event.
- **Cost**: A numeric value used to measure the resources consumed as a result of a particular Event.
- **Duration**: A numeric value used to measure the time it takes to complete an Event.
- **Enables**: A relationship where an Event can allow the execution of another Event.
- **Disables**: A relationship where an Event can disallow the execution of another Event.
- **Hinders**: A relationship where an Event can negatively affect the Cost, Quality, and Duration of another Event.
- **Facilitates**: A relationship where an Event can positively affect the Cost, Quality, and Duration of another Event.
- **Task Tree**: A way to represent Events and the relationships between them.
- **Tick**: A one unit advancement of an internal counter used as a clock.

Chapter 3

General System Requirements

The MASS is designed as a domain independent central process where all domain knowledge is obtained from the SFI and CFI. To prevent any interference or assumption, the MASS and the Agents run as separate independent processes. The cTAEMS model supplies the internal knowledge used to built a Task Tree from which Agents will select available Events for execution. This coupled with the easy configuration mechanism, logging, reporting and statistical analysis makes the simulator a good platform for research and evaluation of Multi-Agent Systems.

3.1 System Features

The following list offers a brief outline and description of the main features and functionalities of the MASS. The features are split into two major categories: core features and optional features. Core features are essential to the application's operation, whereas optional features are preferred but not required. For a better understanding of the basic features of the system see Figure 3.1.

Core Features

1. **Running The MASS**: The MASS will run via the command line inside of a BASH script that takes as inputs the SFI and optionally the CFI.
2. **Simulation File Input (SFI)**: The SFI is a cTAEMS file.
   (a) The system is only required to support the use of the AND, OR, and SUM logical functions of the CTAEMS grammar.
3. **Configuration File Input (CFI)**: The program by default will look for a plain text configuration file in the current directory. Optionally, a user can specify the location of this CFI.
   (a) If no configuration file is found the system will make one in the current directory with default values.
   (b) The CFI will contain a random number seed and output file destination.

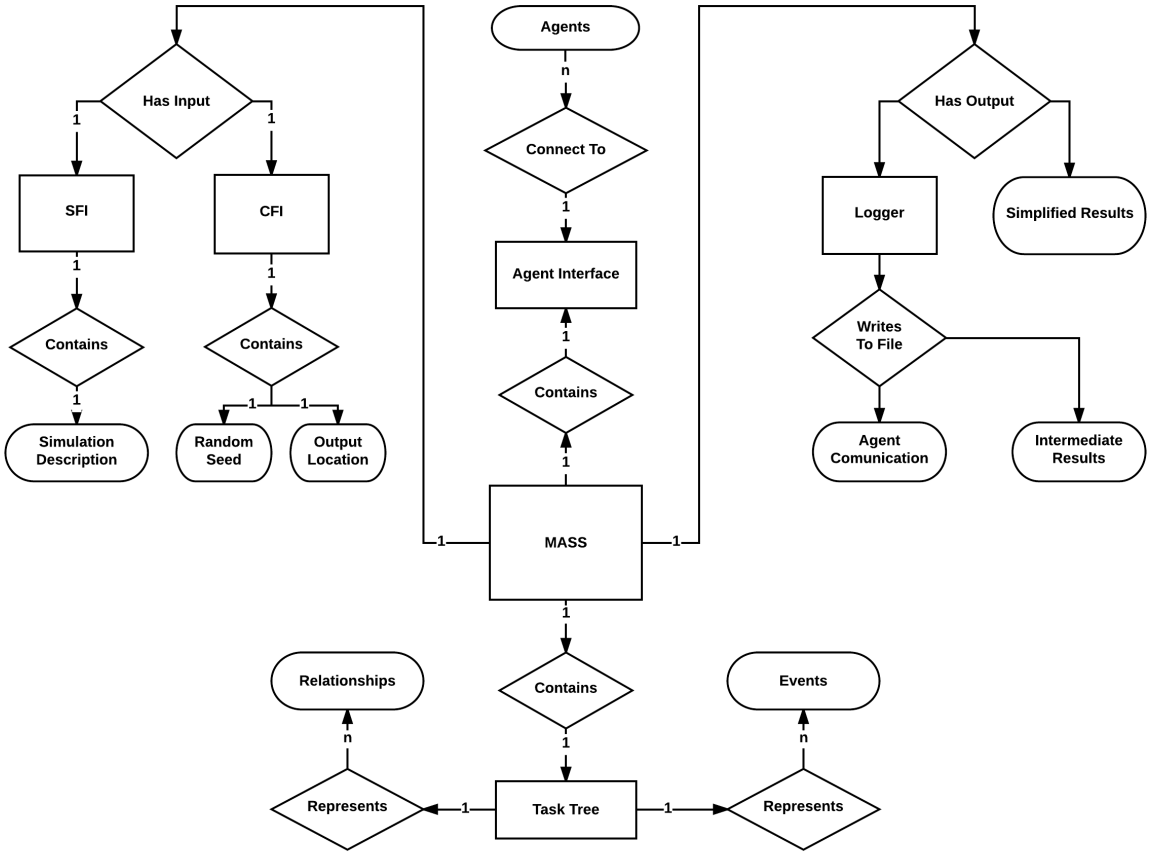

Fig. 3.1: ERD diagram for basic components of MASS.

4. **Agent Facing Interface**: The system will provide a consistent interface of which all Agents must implement in order to connect to the internal communication model.

   (a) All communication between the Agents and the MASS will be represented in the JSON format.

   (b) The system must handle any Agent architecture in compliance with the Agent Interface and Simulation structure and refuse connection to any Agent in violation of these elements.

5. **Agent Communication**: Agents Must have the ability to communicate throughout the Simulation.

   (a) An Agent can send a message directly to another Agent.

   (b) An Agent can broadcast a message to multiple Agents without explicitly knowing the number of Agents that will receive the message.

6. **Simulation Structure**: A Simulation will consist of a Task Tree that keeps track of what Events are able to be executed at any given time.
   (a) It is the job of the agent to choose from the available Events and report to the Simulation when starting or finishing an event.
7. **Creating a Simulation**: A Simulation must be repeatable.
   (a) Any probability distributions must be computed using the seed value obtained from the CFI before the Simulation begins.
   (b) The Task Tree must be built prior to the Simulation?s start.
   (c) The MASS must determine what Events within the Task Tree have no dependencies and pass this information to the Agents prior to the Simulation?s start.
8. **Running a Simulation**: The MASS must keep track of Events.
   (a) The Mass must know what events are able to be executed at any given time.
   (b) The Mass must know what events are being executed at any given time.
   (c) The Mass must know what events have been executed at any given time.
9. **Simulation Output**: The system will produce a command line output detailing the overall Quality, Cost and Duration of the entire Simulation.
   (a) This output should be clear and concise.
10. **Log File Output (LFO)**: The system will produce a detailed log file including the intermediate Quality and Cost between every Task, Subtask, and Method.
   (a) The LFO will contain a transcript of Agent communication
   (b) The LFO will contain compiled statistics on Agent communication frequency.
   (c) The LFO will contain the intermediate and final Quality, Cost, and Duration that results from completing an Event in the Task Tree.

Additional Features

1. **Graphical Representation of Task Tree**: The Task Tree is a hierarchical tree-like structure representing Events and the relationship between them.
   (a) The system may represent this structure as a graphical model to help understand the Simulation results better.
2. **CTAEMS Grammar Support**: The system may go beyond the logical AND, OR, and SUM operations and include additional implementations within the cTAEMS grammar. Figure 3.2 represents a code snippet of CTAMES grammar.

```
(spec_agent
   (label ted)
   (location origin)
)

(spec_location
   (label origin)
   (latitude 12.0)
   (longitude 43.0)
)

(spec_task_group
   (label head)
   (subtasks left_task right_task )
   (qaf q_sum)
)
```

Fig. 3.2: CTAMES grammar

3. **Agent Behavior**: The System may support the ability of Agents to stop, pause, or resume Tasks, Subtasks, and Methods if such actions are applicable based on the Simulation specifications.

Chapter 4

Other non-functional requirements

4.1 Performance requirements

All Agents involved with the model are connected to the simulator using sockets. The Agents themselves are independent processes, which could run on physically different machines. Note that the simulator does not control the Agents' activities, it merely allocates time slices and records the Events performed by the Agent during the time slice. This makes the performance of the MASS independent of an Agents performance. The system shall function in real-time, however, the effect of network load or maximum limit on socket connections can only be published after testing.

4.2 Maintainability

The standardized design and implementation documents will be provided in order to maintain the system. All changes will be documented. A standard architecture will be applied and therefore allowing for quick evolution of the software to adapt to possible situations in the future.

4.3 Software Quality Attributes

The user interface of the Multi-Agent System Simulator is to be designed with usability as the first priority. The system will be presented and organized in a manner that is both visually appealing and easy for the user to navigate. To ensure reliability and correctness, there will be zero tolerance for errors in the Simulation environment.

Chapter 5

Detailed System Description

5.1 Design overview

The system is broken into five major packages called application, input, output, simulation, and tasktree. Figure 5.1 shows the relationship between these components.

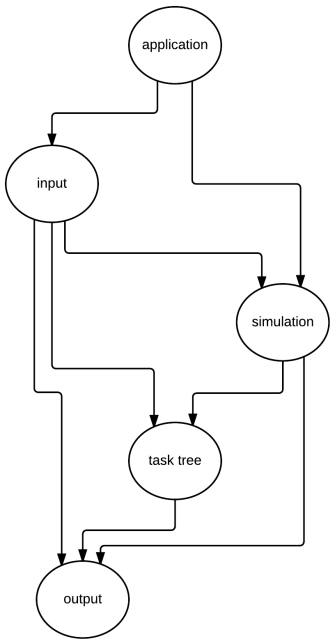

Fig. 5.1: Uses diagram for packages within MASS.

5.2 Package Overview

1. **application**
    - Description: The application package is responsible for initializing the other packages.

- Contains: The application package contains module AISim.
- Uses: The application package does not use any other packages.

2. **input**
   - Description: The input package is responsible for reading the initial data input and building a Task Tree to represent this data.
   - Contains: The input package contains three modules which include Parser, ConfigurationData, and InputData.
   - Uses: This package uses the tasktree package.
3. **output**
   - Description: The Output Package is responsible for logging data after the simulation has completed.
   - Contains: The output package contains a module Logger.
   - Uses: This package uses the simulation package
4. **simulation**
   - Description : The simulation package is responsible for connecting and communicating with agents, managing the tasktree, and advancing the clock.
   - Contains : This package contains the modules Simulator, Agent, EventManager, Message and Thread.
   - Uses : This package uses the input Package.
5. **tasktree**
   - Description : The tasktree package is responsible for representing the Task Tree in a hierarchical manner that is easy for the simulation package to update and distribute. Figure 5.2 represents the Uses diagram for the package tasktree.
   - Contains : This package contains the modules Node, NodeRelationship, and Distribution.
   - Uses : This package uses the output package.

5.3 Detailed Package Design

1. **Application** The application package is responsible for initializing the modules within the input and simulation packages in order to begin the Simulation.

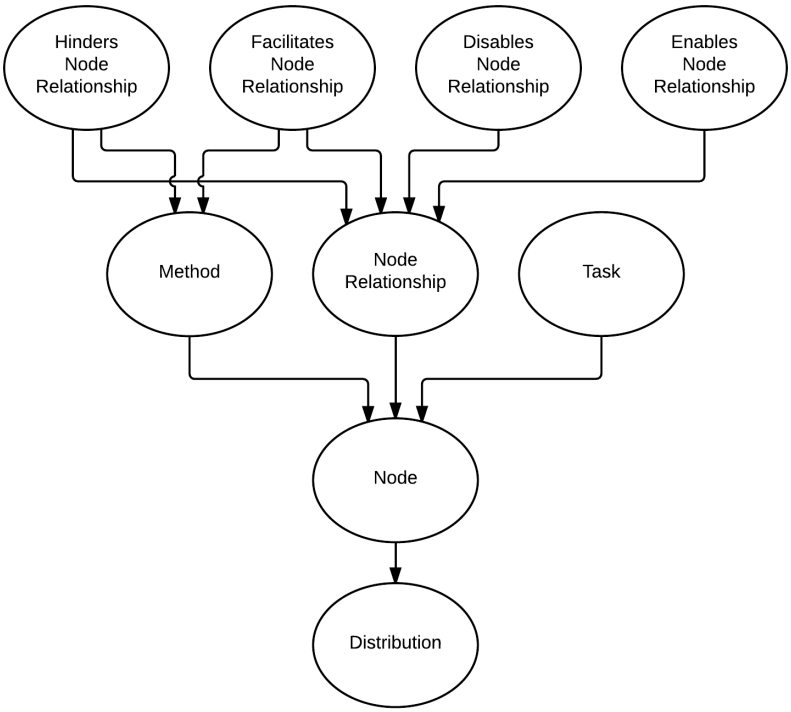

Fig. 5.2: Uses diagram for the package tasktree

2. **Input** Figure 5.3 represents the UML diagram for input package. This package is responsible for reading and parsing the cTAEMS SFI and the plain text CFI. This packages contains an abstract parser which serves as a template for the InputParser and the ConfigurationParser.

   - **InputParser**: The InputParser is responsible for reading the SFI which contains definitions for Agents, Events and relationships that make up the Simulation. The first step of the parse involves reading the desired cTAEMS file into a string by implementing Java's BufferedReader class. Second, the string is split into tokens such as Agent, Task, TaskGroup, and Method. These tokens are further parsed in order to analyze their various components. For example, a Task could be broken up into a label, list of Subtasks and a function of Quality which are instantiated into the corresponding data structure. This data will be stored in the InputData class which will then be passed to the simulation.

   - **ConfigurationParser**: The configuration parser reads and parses the CFI to determine the random seed used throughout the Simulation. The Configuration parser will use Java's BufferedReader class to read in the file, with

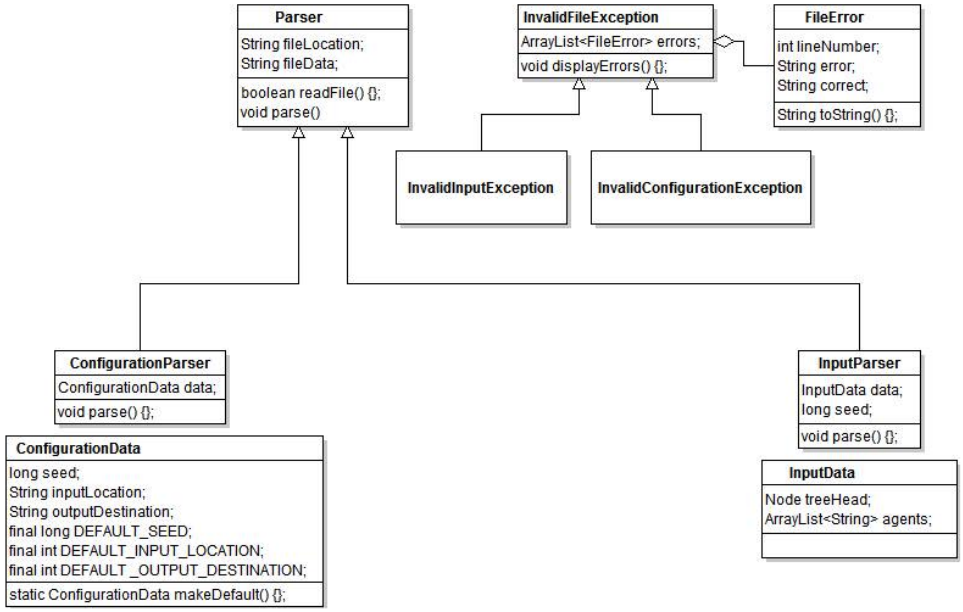

Fig. 5.3: UML diagram for input package

the first line representing the seed, the second the input location and lastly the output destination. This data is written to the ConfigurationData class which is sent to the simulator.

- **Exceptions**: Within the input package there are three exceptions which are used to report a problem encountered during a parse. These exceptions are: InvalidFileException, InvalidInputException, and InvalidConfigurationException.

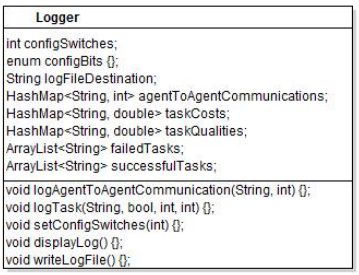

Fig. 5.4: UML diagram for output package

3. **Output** Figure 5.4 represents the UML diagram for output package. The output package is responsible for logging data after the simulation has completed. It contains a Logger class which upon completion will create the LFO document.

- **Logger**: The Logger class is responsible for producing a transcript of Agent communication, statistics on Agent communication frequency, and the intermediate and final Quality, Cost, and Duration that results from completing an Event in the Task Tree. Data will be passed to the Logger by the modules within the simulation package as the simulation advances. Upon completion of a simulation, the Logger will compile statistics on agent communication and produce the LFO containing the previously mentioned data.

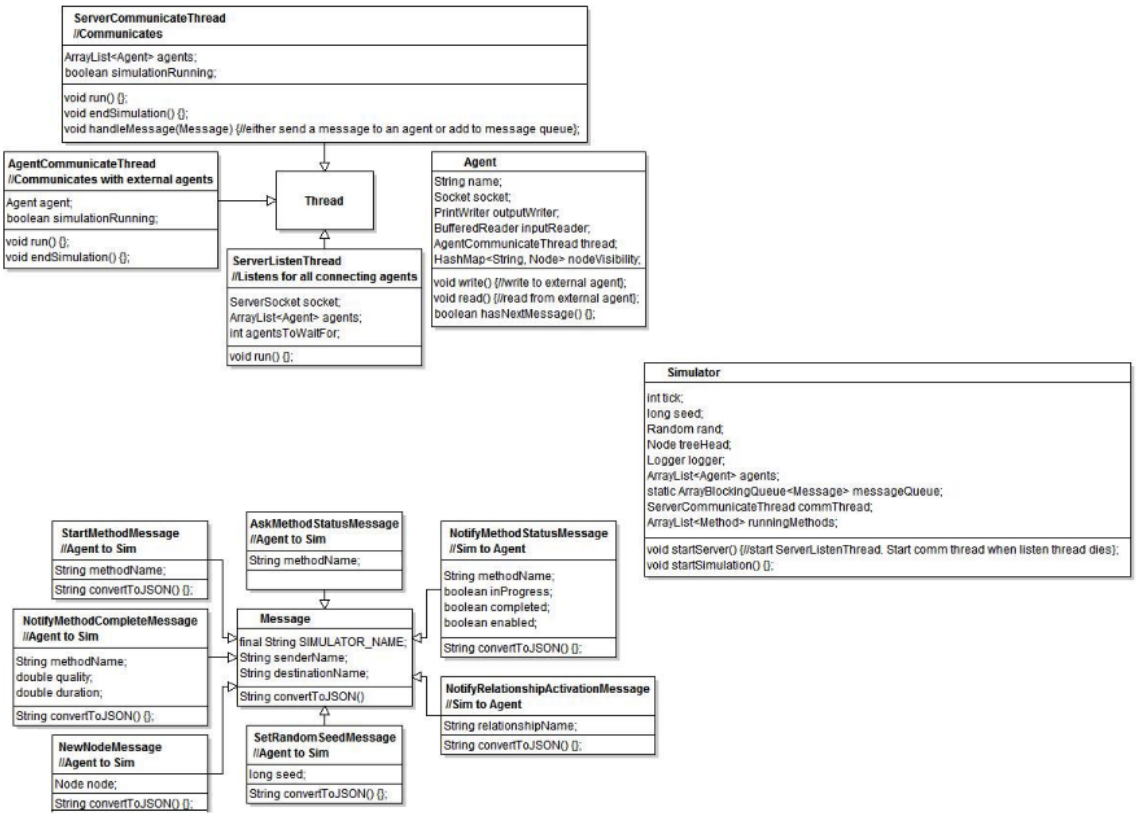

Fig. 5.5: UML diagram for Simulation package

4. **Simulation** Figure 5.5 represents the UML diagram for simulation package. The simulation package is responsible for handling all of the communications with Agents as well as updating the tasktree architecture to model the current state of the simulation.

- **Simulator**: This class contains all of the information about the simulation such as the random number seed, the head Node of the tasktree, the list of currently connected Agents, the Logger object, the incoming Message queue,

the list of currently running Methods, and the ServerCommunicateThread. This class is the core timekeeper of the simulation.

- **Agent**: This class is used to encapsulate all the relevant information about an Agent such as its name, Socket, and HashMap of visible Nodes. This class also contains a PrintWriter, BufferedReader, and AgentCommunicateThread used to communicate with the Agent program that it represents.

- **ServerListenThread**: This Thread will listen for new Agents that are trying to connect to the simulation. If an Agent connects at the correct time and with the correct name then this thread packages the Agent's information into an Agent object so that it can be accessed and communicated with throughout the simulation.

- **ServerCommunicateThread**: This Thread will create and process all Messages comming in from and going out to all Agents in the simulation.

- **AgentCommunicateThread**: This Thread will send and recieve Messages to and from a given Agent.

- **Message**: This class contains the base information for a Message such as the name of the Simulator, the senderName, and the destinationName. There are many subtypes of Messages listed below.

  (i) StartMethodMessage: Tells the Simulator that the Agent wants to start a Method.

  (ii) AskMethodStatusMessage: Ask the Simulator for the status of a currently running Method.

  (iii) NotifyMethodStatusMessage: Tells an Agent the status of a currently running Method.

  (iv) NotifyMethodCompleteMessage: Tells an Agent that their Method has completed and what the result is.

  (v) NewNodeMessage: An Agent tells the Simulator to add a Node to the tasktree.

  (vi) SetRandomSeedMessage: Tells an Agent to set their random number seed to a specified number.

  (vii) NotifyRelationshipActivationMessage: Tells an Agent that a NodeRelationship was activated and what the result is.

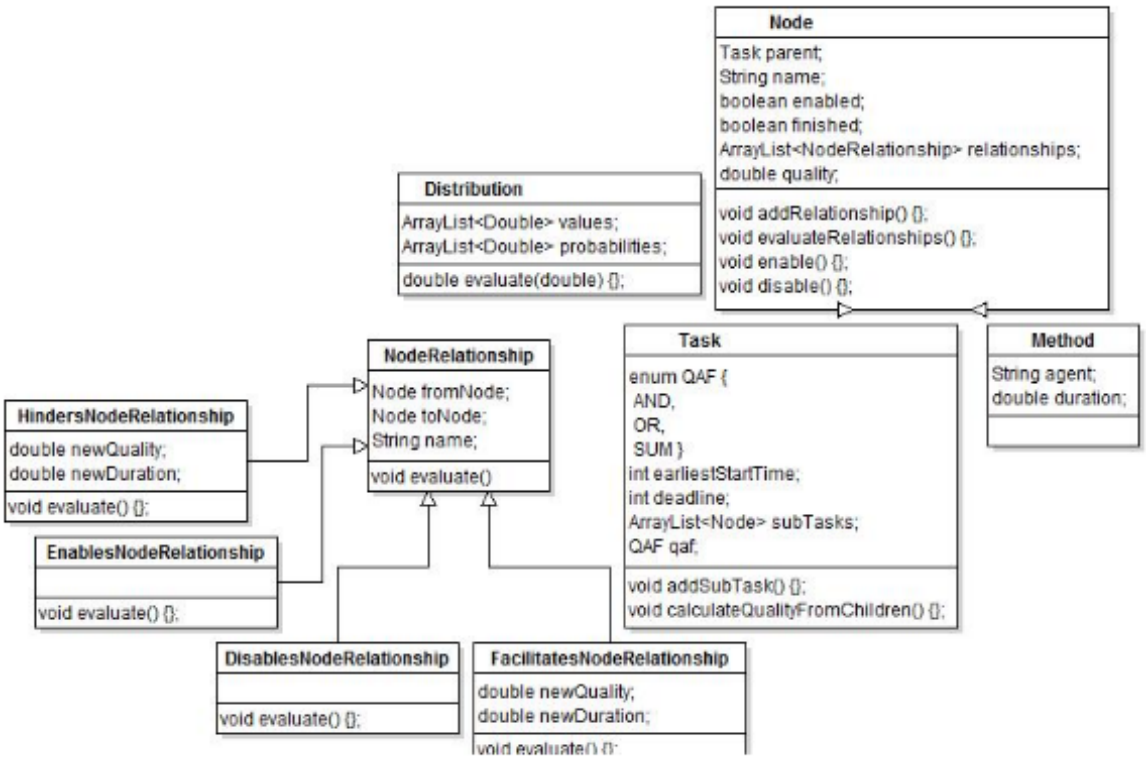

Fig. 5.6: UML diagram for tasktree package

5. **Tasktree** This package provides a framework for building a Task Tree. It is used by the input package to construct a programmatic representation of the SFI including Task, Methods, and Node relations.

   - **Node**: The base node class that contains all the information about a Node of the tasktree such as Quality, name, this Nodes parent Node, a list of NodeRelationships from this Node, whether or not the Node is enabled, and whether or not the Node is finished.

   - **Task**: A Task is a type of node whose completion is defined by its Subtasks. A Task's completion quality is the result of its subtasks. Either the Sum of the Quality of the Subtasks, the highest Quality of any Subtask, or the lowest Quality of any Subtask. More Quality options can be added as needed. A Subtask can either be another Task which would have more Subtasks or a Method which has no Subtasks.

   - **Method**: A Method is a node that can be completed by an Agent. The Method contains a Duration which is the amount of time it takes to com-

plete the Method. Methods can have their Duration and Quality modified by the FacilitatesNodeRelation and HindersNodeRelation.

- **NodeRelations**: There are many NodeRelations that represent the affect that the completion of one Node has on another. The types of NodeRelations are below.

  (i) HindersNodeRelation: Completing one Node decreases the Quality and increases the Duration of another Node.

  (ii) FacilitatesNodeRelation: ompleting one Node increases the Quality and decreases the Duration of another Node.

  (iii) EnablesNodeRelation: The completion of the first Node enables the completion of the second Node

  (iv) DisablesNodeRelation: The completion of the first Node disables the completion of the second Node.

- **Distribution** A distribution is used to generate Qualities and Durations for the Nodes.

Chapter 6

Conclusion

Overall, a comprehensive list of requirements has been created. Due to the explorative and innovative character of the project, it is rather a broad and deep insight in the examined area of Multi-Agent System Simulator. Thus, it should facilitate further focusing on project aims and guide to successful case studies and prototypes. The latter will be used to identify new or still overseen demands.