HyCube Simulator documentation

v.1.0.1

Author: Artur Olszak

09.10.2016

Contents

1. Simulator architecture

The implemented simulator consists of two main components:

- **Simulator**
- **Simulator master**

The simulator component is a container running multiple node instances, providing an interface for performing operations on the simulator instance and on individual nodes. The operations include creating new nodes (joining the DHT), dropping nodes (simulating failures) and running operations in individual nodes' contexts (sending messages, performing lookup, search and the DHT operations, as well as calling nodes' extensions, background processes, running the recovery procedure and leaving the system). The simulator also maintains a number of counters: the total number of messages sent, the number of messages delivered/lost, and the route lengths (using an additional received message processor and message send processor (*StatReceivedMessageProcessor*, *StatMessageSendProcessor*), which collect information about messages being sent and received).

Multiple simulator instances may be connected to each other, and the nodes maintained by individual simulators may form a common DHT system, using virtual network addresses, determining on which simulator instance nodes are located, and identifying the nodes within individual simulators. The communication method (virtual network layer) between nodes located within the same simulator instance, as well as located in separate simulator instances, is defined by implementing a simulator-specific network proxy.

Figure 1.1 presents the architecture of the simulator. The central object is the simulator master (an instance of the *SimulatorMaster* class). The simulator master provides methods establishing connections to individual simulator instances and performing operations on them, using an internal simulator service proxy object (implementing the interface *SimulatorServiceProxy*) and an appropriate service (stub) passing requests to the simulator object. The simulator instances are identified by the simulator ID and the connection information is determined by the connection string provided to the simulator

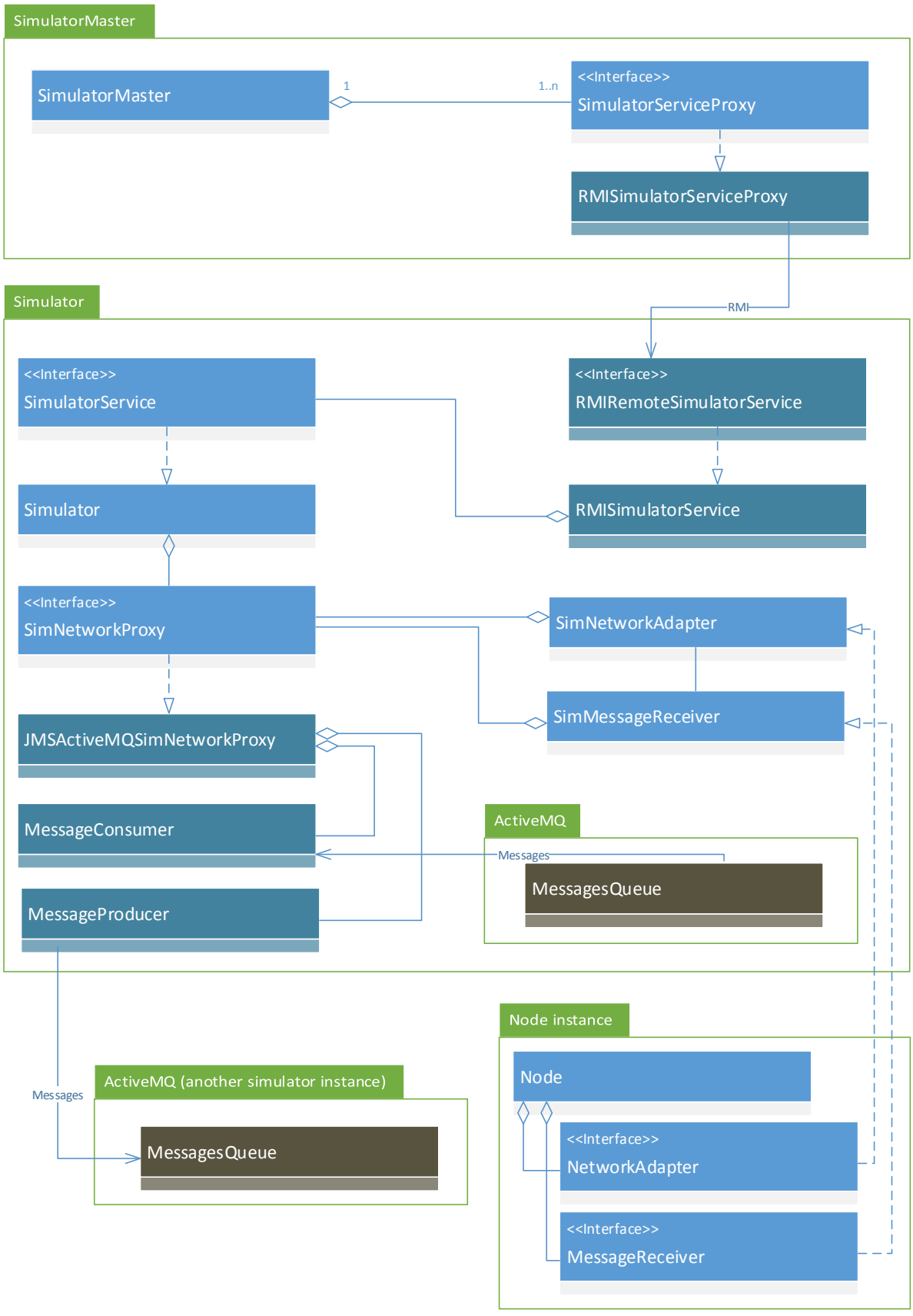

Figure 1.1 Simulator architecture

master (the interpretation is *SimulatorServiceProxy* implementation dependent). The simulator service proxy object is created by the simulator master through a factory object (*SimulatorServiceProxyFactory*), passed to the *initialize* method of *SimulatorMaster*. The proxy object connects to the simulator (an instance of *Simulator* class implementing the *SimulatorService* interface) through a stub object (on the simulator side), which maintains a reference to the simulator instance and passes requests to the simulator object.

A message being sent is passed by node's network adapter to the simulator network proxy object, maintained by every simulator (implementing *SimNetworkProxy* interface), which transfers the message to the appropriate simulator instance (the communication method is determined by the network proxy implementation). The message receiver of the receiving simulator also retrieves the message using the abstraction of the network proxy, and upon receiving the message, passes it to the network adapter of the appropriate node. The simulator network proxy instance is defined upon the simulator object initialization and is responsible for establishing connections between simulator instances, as well as for sending messages to nodes and receiving messages - the network proxy object is bound to simulator network adapter and simulator message receiver instances - *SimNetworkAdapter* and *SimMessageReceiver* - the network adapter and message receiver implementations that use the simulator network proxy object for sending and receiving messages. The simulator network proxy implementation defines the network layer (how simulated nodes communicate with each other), and the simulator logic is independent of this implementation. The node network address is defined by *SimNetworkAdapter* and *SimMessageReceiver* as a string in the following format: *SIMULATOR_ID:NODE_SIM_ID*, where *SIMULATOR_ID* is the simulator ID (4-character ISO-8859-1 string) and the *NODE_SIM_ID* (16-character ISO-8859-1 string) is the simulation identifier of the node. The simulator ID is specified when the simulator object is created, and the simulation node ID is specified when the node is created (passed through the simulator service proxy to the simulator instance). Based on the simulator ID, the simulator network proxy of the sending node (simulator instance maintaining the sending node) should be able to determine to which simulator instance messages should be transferred, and, upon receiving a message, the *SimMessageReceiver* instance (of the simulator maintaining the recipient) is responsible for determining the node (network adapter instance) to which the message should be passed. *SimNetworkAdapter* and *SimMessageReceiver* operate on network addresses represented by *SimNodePointer* class (implementing *NetworkNodePointer*), and the

binary representation of the address is the concatenation of byte representations of simulator id and simulation node id, encoded in ISO-8859-1 (20 bytes long). Connections between individual simulators are established by calling the simulator master's methods, specifying simulator network proxy specific connection strings, defining the connections. The simulator master passes connection requests to individual simulators, which pass the requests to their simulator network proxy objects, responsible for establishing and maintaining connections (the simulator master uses the network proxy abstraction, the connection realization is network proxy implementation dependent).

The simulator service proxy implemented in the simulator is based on the Java RMI technology and is represented by class *RMISimulatorServiceProxy* (implementing *RMISimulatorSerevicеProxy*). The factory object passed to the simulator master initializer is an instance of class *RMISimulatorServiceProxyFactory*. The RMI proxy connects to the stub object of class *RMISimulatorService*, implementing the remote interface *RMIRemoteSimulatorService*. The *RMISimulatorService* object internally maintains a reference to an object implementing the *SimulatorService* interface (the simulator), passed to the class constructor upon creation. The stub object simply passes the calls received to the simulator object. The connection string format expected by *RMISimulatorSerevicеProxy* is the standard URL format defining RMI remote objects: *//host:port/name*.

The simulator network proxy implementation provided with the library is based on the JMS queues (ActiveMQ implementation). The network proxy object maintained by the simulator object is an instance of class *JMSActiveMQSimNetworkProxy* (implementing *SimNetworkProxy*). Instances of the network proxies establish connections to the local ActiveMQ queues (consumer), and to the queues of all connected (including self) simulator instances (as a producer). Whenever a message is sent, the node proxy object sends it to the ActiveMQ queue (packed in the instance of the *SimMessage* class) of the appropriate simulator, and message receiver instances, through the abstraction of the network proxy, receive the messages from the local ActiveMQ queues. The connection string format expected by *JMSActiveMQSimNetworkProxy* is: *protocol://host:port[queue]*, where *protocol*, *host* and *port* define the JMS connection, and *queue* is the message queue name, for example: *tcp://testmachine01:51001[SimMessages]*.

The simulator application (service) - class *SimulatorRemoteServiceApp1* creates an instance of *JMSActiveMQSimWakeableNetworkProxy*, initializes a *Simulator* instance and creates an

instance of *RMISimulatorService*, passing the reference to the simulator to it and exposing the remote object through RMI.

The master application (*SimulatorMasterApp1* class) creates a simulator master (*SimulatorMaster* class) instance and runs a simulation script - a class implementing the *Simulation* interface specified as a command line argument (by calling its *runSimulation* method, passing the simulator master object and the remaining command line parameters to it). The simulation script (implementation of the *Simulation* interface) may connect to individual simulator instances and realize any possible simulation scenarios.

2. Changelog

1.0

- Initial release

1.0.1

- Changed the build automation to Maven
- Integration with Travis CI