Design Document

Ben Soer & Eric Tsang, 6D

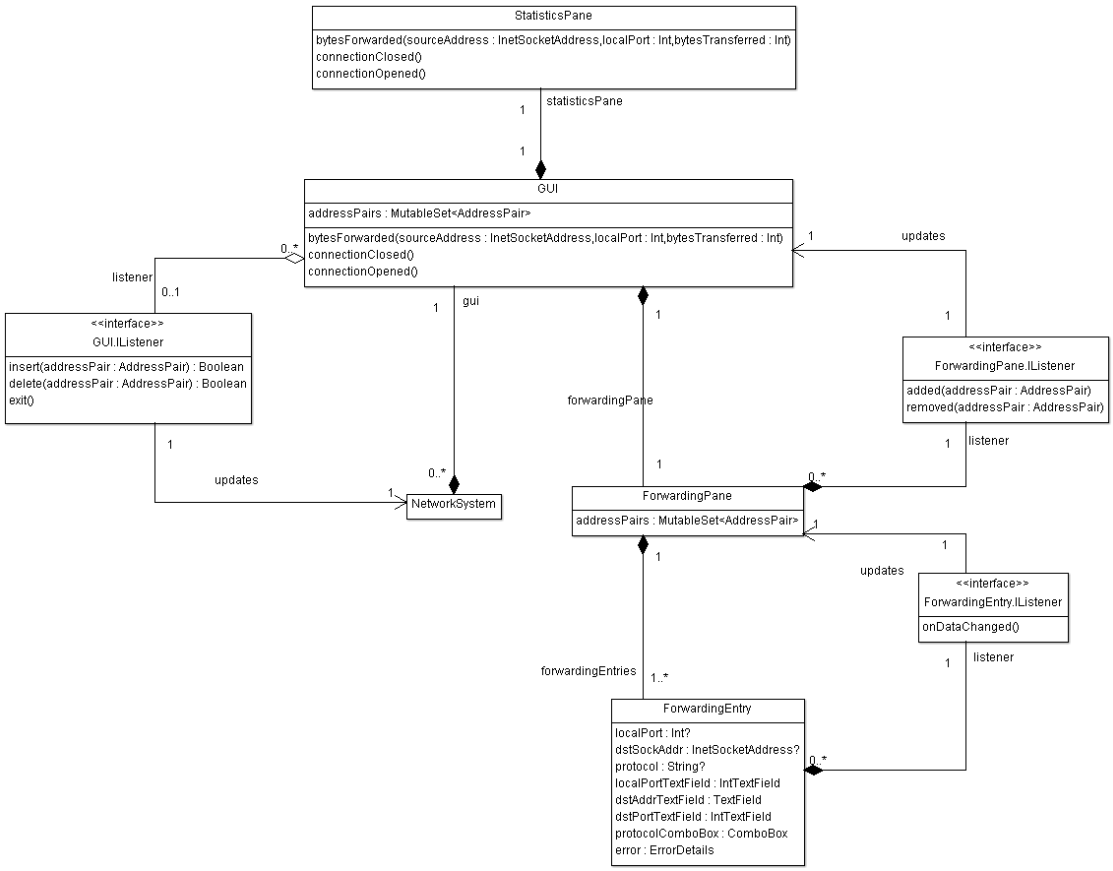
1 Table of Contents

2 Class Diagrams

This section contains class diagrams describing the various classes of the GUI and networking modules.

2.1 Graphical User Interface

Below is a class diagram of all the main classes in the GUI module.

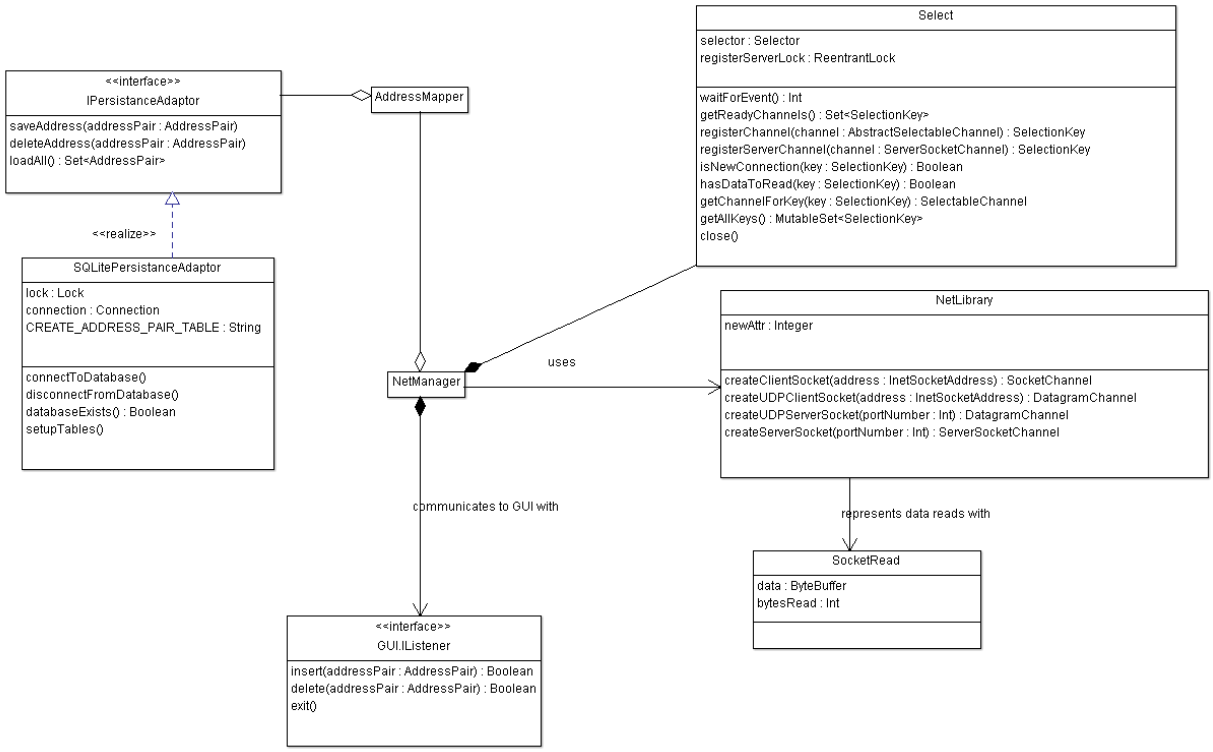


Name	Description
NetworkSystem	Represents the networking module of the application. It directly interacts with the `GUI` object to communicate statistics to it, and add and remove entries to and from the `GUI`. It also registers a `GUI.IListener` object with the `GUI` so it may be notified of events that occur to the `GUI`, like when the user inputs forwarding information into the `GUI`, and when it exits.
GUI	The `GUI` class manages the displayed window, and the `StatisticsPane` and `ForwardingPane` within it. It acts as a facade to the `StatisticsPand` and `ForwardingPane` within it. The members of the `GUI` are aliases of those in its `ForwardingPane` and `StatisticsPane`. It registers a `ForwardingPane.IListener` with the `ForwardingPane`, so whenever forwarding entries are added or

	removed, it may notify all `GUI.IListeners`, and display the proper error dialogs if things go wrong.
GUI.IListener	Interface that GUI observers should implement. The implemented methods are callbacks about high level events that occur on the GUI.
ForwardingPane	This can be seen as the top pane on the `GUI`. It contains one or more `ForwardingEntry` objects, and manages them. It makes sure that duplicate ports are shown in red, and makes sure that there is at least one empty forwarding entry, so users may add more forwarding entries as they please. Registers a `ForwardingEntry.IListener` with all `ForwardingEntry` objects, so it may evaluate all inputted forwarding entries for duplicate ports, and flag them as errors.
ForwardingPane.IListener	Interface that `ForwardingPane` observers must implement to be notified about the addition of new, or removal of old `AddressPair`s to and from the `ForwardingPane` by the user.
ForwardingEntry	Manages the various input components, and makes sure that they are put in the correct formats. Allows a user to specify a forwarding mapping; which port to forward received packets from, the destination address and port to forward the packets to, and the protocol to forward.
ForwardingEntry.IListener	Interface that `ForwardingEntry` observers must implement. observers are notified whenever the state of the forwarding entry changes.
StatisticsPane	This can be seen as the bottom pane on the `GUI`. It keeps track of all statistics, and manages the statistic displays.

2.2 Networking

Below is a class diagram of all the main classes in the Networking module.

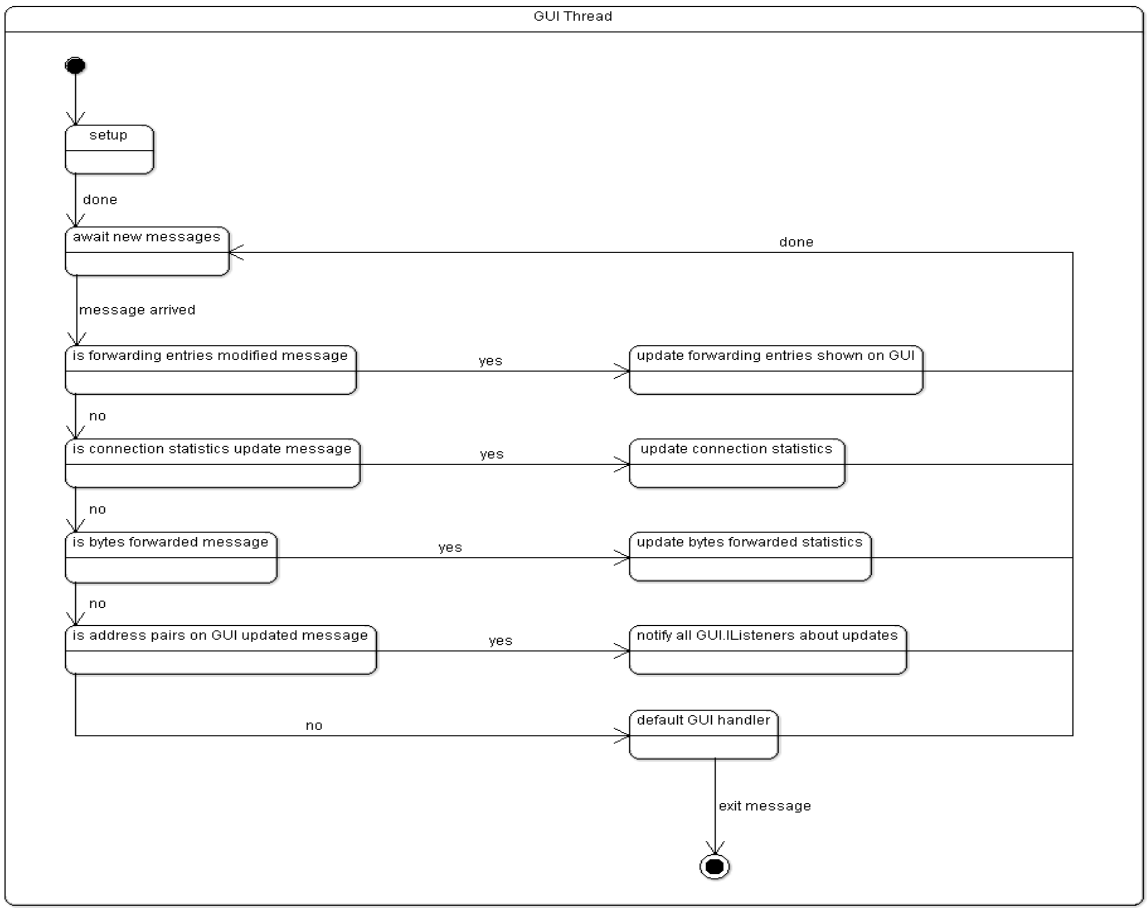


Name	Description
NetManager	The `NetManager` is used to couple all the class into the thread
AddressMapper	Associates ports on the forwarding machine with the user-specified destination address and port. Takes an `IPersistenceAdapter` to persist entered data, so it can reload previous configurations when restarting the application.
IPersistenceAdapter	Interface that persistence strategies must implement.
SQLitePersistenceAdapter	A persistence strategy that implements `IPersistenceAdapter`. Uses SQLite to persist data.
NetLibrary	Contains helper functions used to create `SocketChannel`s and `DatagramChannel`s.
SocketRead	Represents data read from a `DatagramChannel` or `SocketChannel`.
Select	Monitors multiple `SocketChannel` or `DatagramChannel` objects. On windows, it maps to the `select` system call. on *nix, it maps to the `epoll` system call.
GUI.IListener	Interface that `GUI` observers should implement. The implemented methods are callbacks about high level events that occur on the `GUI`.

3 State Diagrams

This section contains finite state machine diagrams describing the various logical states that threads of the GUI and networking modules go through.

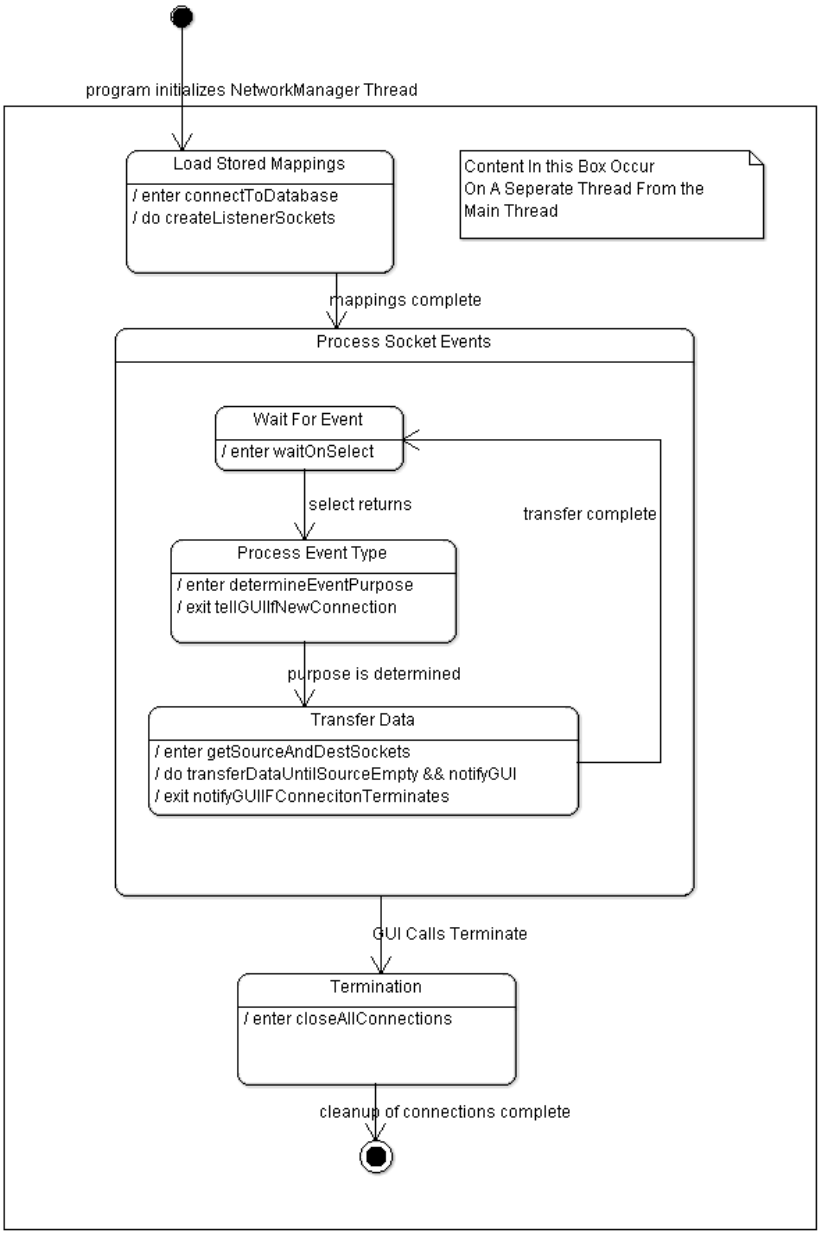
3.1 Graphical User Interface



Name	Description	Transitions
Setup	Initializes data, adds panes to the windoe, and sets up the GUI.	**Done:** the operation was completed.
Await new messages	Blocks until a message arrives in the message queue to handle.	**Message arrived:** a new message arrives and can be de-queued from the message queue.
Is "forwarding entries modified" message	Checks if the received message indicates that the collection of forwarding entries have been modified.	**Yes** if the message is a "forwarding entries modified" message; **no** otherwise.
Update forwarding entries shown on GUI	Adds missing `AddressPair` objects from the internal collection to the `ForwardingPane`	**Done:** the operation was completed.

	on the `GUI`, and removes extra ones.	
Is "connection statistics update" message	Checks if the received message indicates that a new connection was created, or an old one was removed, and that the statistics about connections must be updated.	**Yes** if the message is a "connection statistics update" message; **no** otherwise.
Update connection statistics	Update the connection statistics as specified in the message, and infer other statistics about connections from the update.	**Done:** the operation was completed.
Is "bytes forwarded" message	Checks if the received message indicates that data was received, and forwarded to the specified server.	**Yes** if the message is a "bytes forwarded" message; **no** otherwise.
Update bytes forwarded statistics	Update bytes forwarded statistics as specified in the message, and infer other statistics about bytes forwarded from the update.	**Done:** the operation was completed.
Is "address pairs on GUI updated" message	Checks if the entries in the `GUI` have been modified by the user.	**Yes** if the message is a "address pairs on GUI updated" message; **no** otherwise.
Notify all GUI listeners about updates	Determines what the changes on the `GUI` were by comparing it with its previous state, and notifies all registered `GUI.IListeners` about all the differences.	**Done:** the operation was completed.
Default GUI handler	Message is handled by the `JavaFX` platform to resize the window, or input text into controls and so on.	**Done:** the operation was completed.

3.2 Networking



Name	Description	Transitions
Load stored mappings	The system loads the previous address mapping configuration from persistent storage, and communicates the loaded address mappings to the `GUI` and opens the appropriate server sockets.	**Mappings complete:** the operation is completed, and all addresses are loaded.
Wait for event	The system waits for activity to occur on any of the open ports and server ports.	**Select returns:** the selector function unblocks, indicating activity on sockets.

Process event type	The system determines what kind of activity has occurred, and sends a message to the `GUI` to update connection statistics if necessary.	**Purpose is determined:** the message type and how to handle it is determined.
Transfer Data	The system forwards all available received data on every connection to their appropriate destinations, then closes all terminated connections, and messages the `GUI` thread of terminated connections as well.	**Transfer is complete:** the operation is completed.
Termination	The termination callback was invoked, and all connections should be closed. All user-entered address mappings should be persisted.	**Cleanup of connections complete:** data has been persisted, and all connections, closed.

4 Pseudocode

This section contains pseudocode of the application.

4.1 Graphical User Interface Thread

1. Set up the GUI, and initialize variables
2. Enter the event loop…await messages
3. Parse received message, and handle it. Handling messages may involve interrupting the networking thread, and sending it messages to perform tasks.
4. Repeat from step 2, unless the message was an exit message

4.2 Networking Thread

1. Load previous from persistent storage, and apply them
2. Await socket activity, or thread interruption
3. Check message queue; setup new server sockets or teardown old ones as necessary; if it is an exit message, terminate all connections, persist configurations, and terminate the thread
4. Assess sockets for activity, and transfer all received data to their destinations
5. Teardown terminated connections
6. repeat from step 2