# The MUX (Multiplexer) Protocol

Wittawat Tantisiriroj, J. Liebeherr, MNG Group

(updated: January 2008)

This is a draft and not a final version.


# Table of Contents

# 1. Introduction

The MUX (Multiplexer) Protocol provides a capability to share the same node adapter among several overlay sub-nodes. To distinguish between messages of the different sub-nodes, the protocol encapsulates a message of a sub-node into the MUX message (see Section 2 Message Format) which contains the index field for demultiplexing the message at the destination.

The goal of the design is to hide the multiplexing operation from an overlay sub-node and a node adapter.

# 2. Message Format

This section list the detailed message formats used in the MUX Protocol. The format for messages is shown in Figure 1.

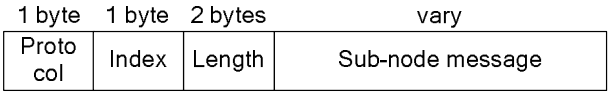


**Figure 1**: MUX Message Format

**Protocol:** The protocol number is one-byte long field that identifies the overlay protocol:
0x50: MUX protocol
**Index**: The index of the sub-node of the sub-node message.
**Length**: The length of the sub-node message of the MUX message in bytes.

# 3. Components of the MUX protocol

## *3.1. MUX Processor*

This component is used to create dummy adapters and to manage the indexes of dummy adapters for the overlay sub-nodes. By using a *createDummyAdapter* function provided by this component (See Section 4.1), an overlay node can create an arbitrary number of dummy adapters which share the same node adapter. However, a message can be exchanged only between the dummy adapters with the same index. This component also performs the demultiplexing operation for incoming message by examining the MUX message for the index and calls a *processMessage* function of the dummy adapter which has the same index as the index in the MUX message (See Section 4.2)

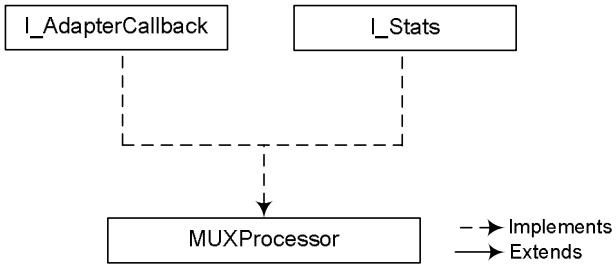


Figure 2: The class structure of *MUXProcessor*

The *MUXProcesssor* implements *I_AdapterCallBack* to handle incoming messages from the adapter. To a node adapter, the *MUXProcesssor* looks the same as the overlay node which implements the

*I_AdapterCallBack* interface. So, the node adapter is not aware of the existence of the *MUXProcessor*. The *MUXProcesssor* also implements *I_Stats* to enhance statistic operations.

The *MUXProcessor* is implemented as the abstract class which will be extended by an overlay node with multiple sub-nodes such as a *Tier-Cluster* node or by an adapter with multiple sub-adapters.

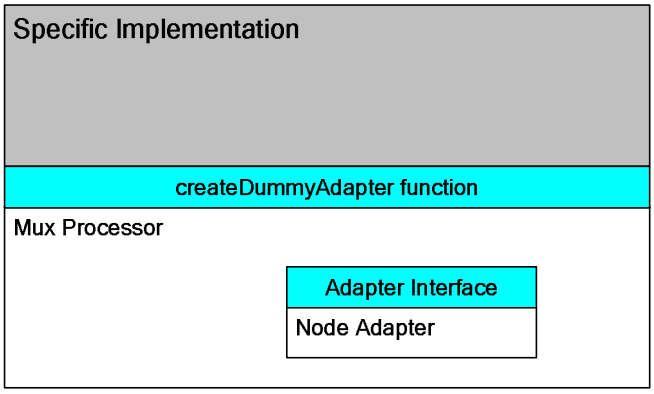


Figure 3: A general object which extends the *MUXProcessor*

The overlay node which extends the MuxProcessor then calls a method createDummyAdapter to a node adapter for each of its sub-nodes before passing these dummy adapters through a method node factory to create sub-nodes.

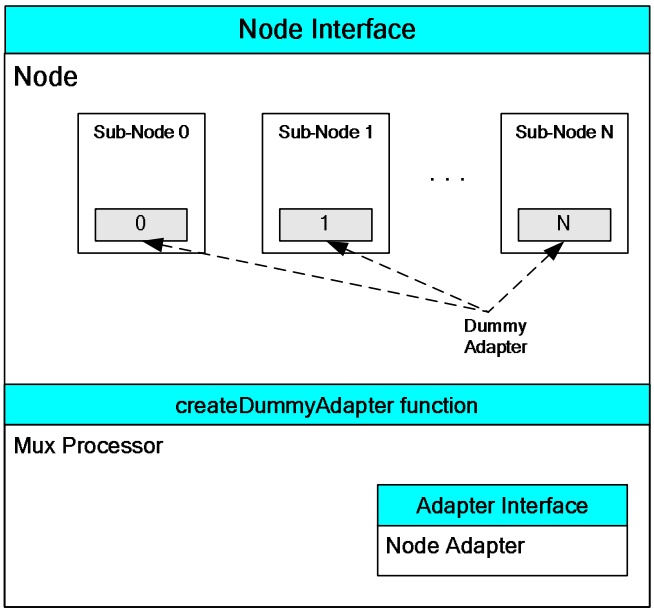


Figure 4: The overlay node which extends the *MUXProcessor*

The adapter which extends the MuxProcessor then also calls a method createDummyAdapter to its sub-adapters. This model may be used to share a single underlay socket between a node adapter and a socket adapter.

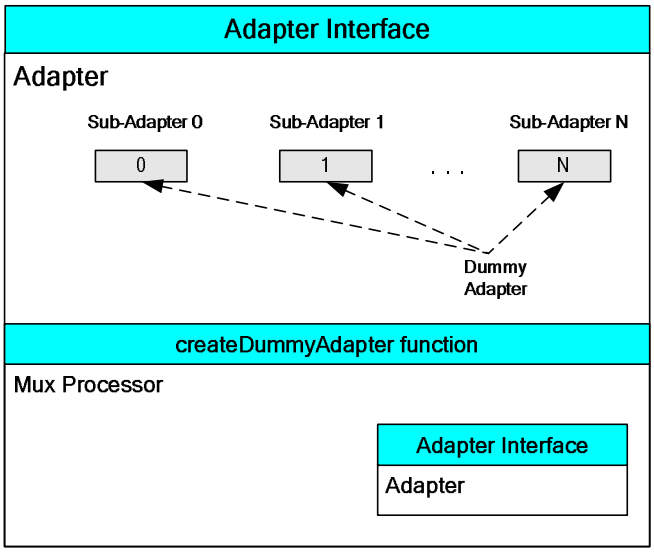


Figure 5: An adapter which extends the *MUXProcessor*

## 3.2. Dummy Adapter

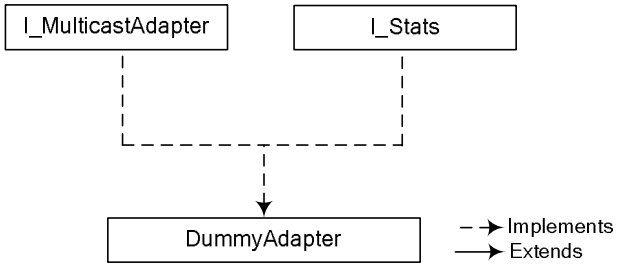


Figure 6: The class structure of *DummyAdapter*

The Dummy Adapter implements *I_MulticastAdapter* to handle outgoing and incoming messages for the sub-node. To a sub-node, the Dummy Adapter looks the same as the regular node adapter which implements the *I_MulticastAdapter* interface. So, the sub-node is not aware of the existence of the Dummy Adapter. Between the MUX Processor and the Dummy Adapter, there is the internal interface to handle all of the adapter operations.

# 4. Interaction

## 4.1. Creation of Dummy Adapters

Before creating a dummy adapter, the MUX Processor must be initialized in order to allocate the table of dummy adapters. An overlay node then calls a function *createDummyAdapter* and provides the index as a parameter. Inside the function *createDummyAdapter*, a new dummy adapter is created with the index and the pointer to the MUX Processor as the parameter in the constructor. So, the dummy adapter has enough information to handle outgoing messages. Then, the MUX Processor registers the dummy adapter to its dummy adapter table so that the MUX Processor has enough information to handle incoming messages.

For example, a dummy adapter is created with the following line of code:

```
//Initialize the MUX Processor in a constructor
Public Object(...){
    super(multicast_adapter);
}
//Create dummy adapter for a sub-node with index of 0
I_MulticastAdapter sub_adapter_0 = super.createDummyAdapter(0);
```

## 4.2. Processing of incoming and outgoing messages

### 4.2.1. Incoming message

When an incoming *MUX message* is received from the node adapter as a byte array, the *MUX message* is reconstruct into a *MUX message* by calling the *restoreMessage* function of the *MUX_Message* class inside the *MUX Processor* (Step 1 in Figure 7). After the *MUX message* arrives from the node adapter, the *MUX Processor* examines the index in the *MUX Message* (Step 2 in Figure 7) and passes only the sub-node message to the correspondent dummy adapter by calling the *processMessage* function of the dummy adapter (Step 3 in Figure 7). Then, an incoming sub-node message as a byte array is reconstruct into a sub-node message inside the sub-node (Step 4 in Figure 7). Finally, the *messageArrivedFromAdapter* function of the sub-node is invoked to process the message (Step 5 in Figure 7).

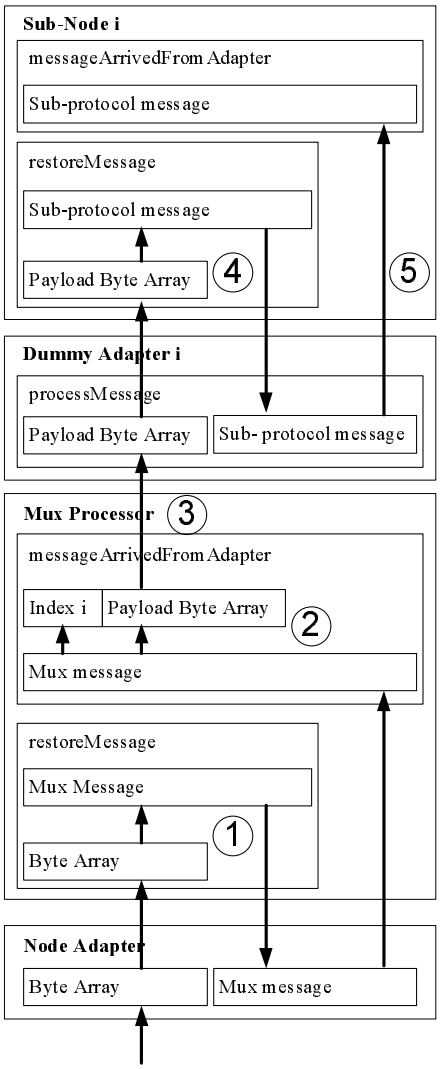


Figure 7: Processing an incoming MUX message.

### 4.2.2. Outgoing message

When the overlay node wants to send a protocol message, it calls the *sendUnicastMessage / sendMulticastMessage* function of its adapter which is the dummy adapter in this case and passes the protocol message as a parameter (Step 1 in Figure 8). The dummy adapter then transforms the protocol message into a byte array and encapsulates it into a *MUX message* which contains the index of the dummy adapter (Step 2 in Figure 8). Then, the dummy adapter calls *mux_sendUnicastMessage / mux_sendMulticastMessage* function of the MUX Processor and the MUX Processor calls *sendUnicastMessage / sendMulticastMessage* function of the node adapter (Step 3-4 in Figure 8). Finally, the node adapter transforms the *MUX message* into a byte array before sending it to the underlay network (Step 5 in Figure 8).

**Comment:** Rename to avoid conflict with I_MulticastAdapter in case MUX Processor is used to multiplex adapter

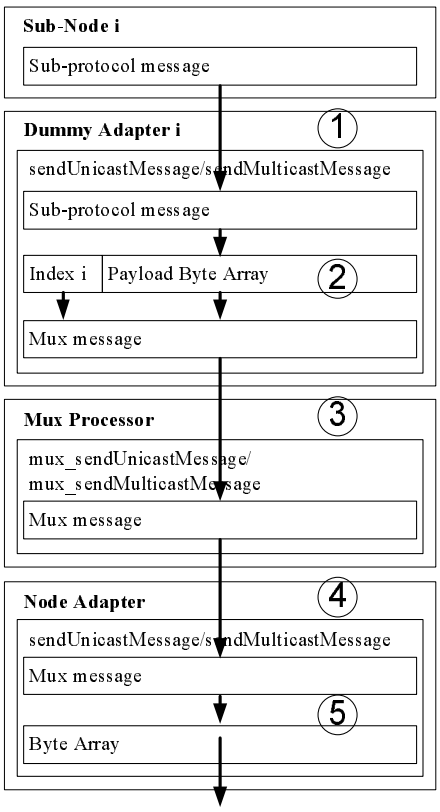


Figure 8: Processing an outgoing MUX message.

## *4.3. Processing of timers*

When the overlay node wants to set a timer, it calls the *setTimer* function of its adapter which is the dummy adapter in this case and passes the timer ID as a parameter (Step 1 in Figure 9). The dummy adapter then encapsulates it into a *MUX Timer ID* which contains the index of the dummy adapter (Step 2 in Figure 9). Then, the dummy adapter calls *mux_setTimer* function of the MUX Processor and the MUX Processor calls *setTimer* function of the node adapter (Step 3-4 in Figure 8). Finally, the node adapter

**Comment:** Same reason as above

uses this MUX Timer ID to create a timer. In case of *getTimer* and *clearTimer* functions, they follow a similar procedure as *setTimer* function does.

When a timer expired, the *timerExpired* function of the *MUX Processor* is called (Step 5 in Figure 9). Then, the *MUX Processor* examines the index in the *MUX Timer ID* (Step 6 in Figure 9) and passes only the sub-node timer ID to the correspondent dummy adapter by calling the *timerExpired* function of the dummy adapter (Step 7 in Figure 9). Then, the dummy called the timeExpired function of the sub-node and the sub-node is invoked to process the timer expired event (Step 8 in Figure 9).

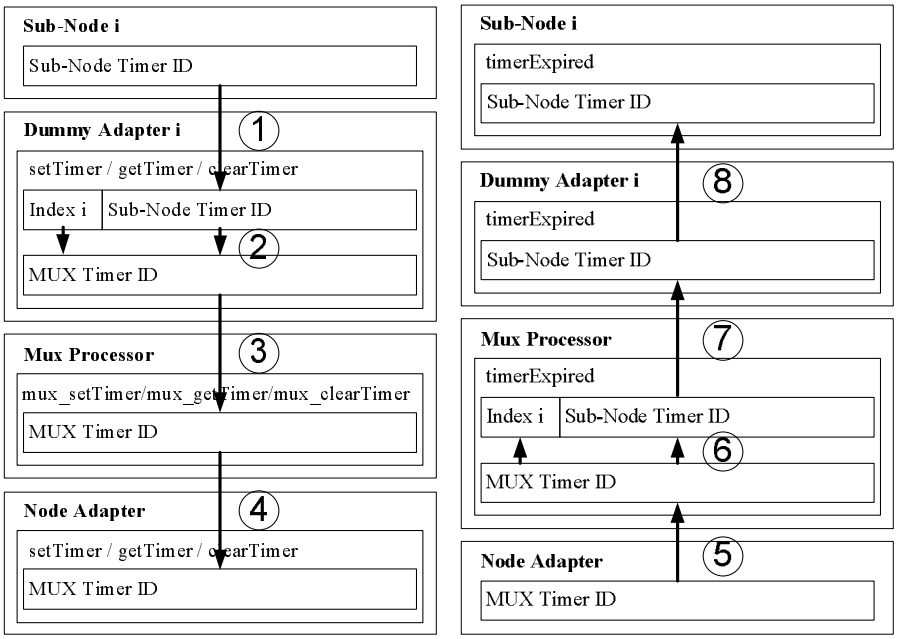

Figure 9: Processing a timer.

## 4.4. Processing of other operations of an adapter

The other operations are straightforward from the Sub-node via the Dummy Adapter via the MUX Processor to the node adapter. For example, the processing of the createPhysicalAddress(String) operation is shown in Figure 10.

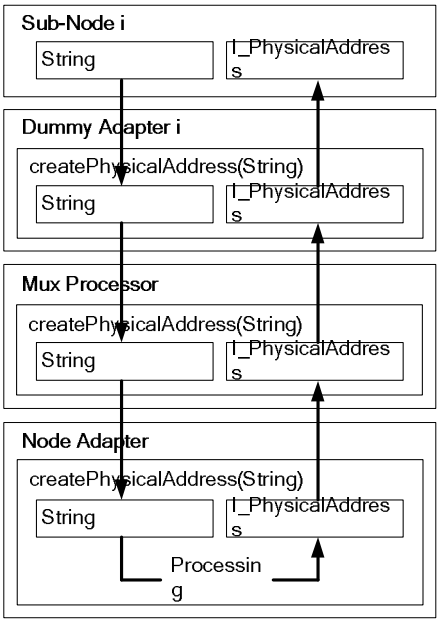

Figure 10: Processing createPhysicalAddress operation.

# 5. Statistics

The MUX Processor support statistics. However, all statistics are hidden behind the statistic interface of the class extended from MUX Processor. A list of supported statistics is following

**Comment:** No clear way to access MUX Processor

- NumOfDummyAdapters
  - The number of the dummy adapter that are registered with a Mux Processor
- NodeAdapter
  - The statistic of underlay adapter that the MUX Processor using.