The Design of a Reliable Broadcast Library for the Readout Control Network

Ichiro Suzuki

June 25, 2001

1 Introduction

We exploit a reliable broadcast protocol, RCNP, on the Readout Control Network (RCN). The current plan considers both FastEthernet and IEEE1394 as the network medium. Therefore, we'd better developping a library, which provides concrete APIs for both Ethernet and IEEE1394 implementations and makes switching between them easy.

The aims of this document are;

1. to formalize the design process,
2. to reach at a concrete final design and
3. to share information among the EVM or the EVB developpers for better understanding of the library.

Each intermidiate design products are shown roughly in chronological order and described in detail to accomplish these goals. This document is not only a report of the final design, but also an example of an object oriented design process.

2 Object analysis

The text description of the requirements is following. The library distributes GTP and event ID information stisfying the requirement to the RCN. It is, however, designed to realize reliable broadcast from a root node of the network to leaf nodes for general data.

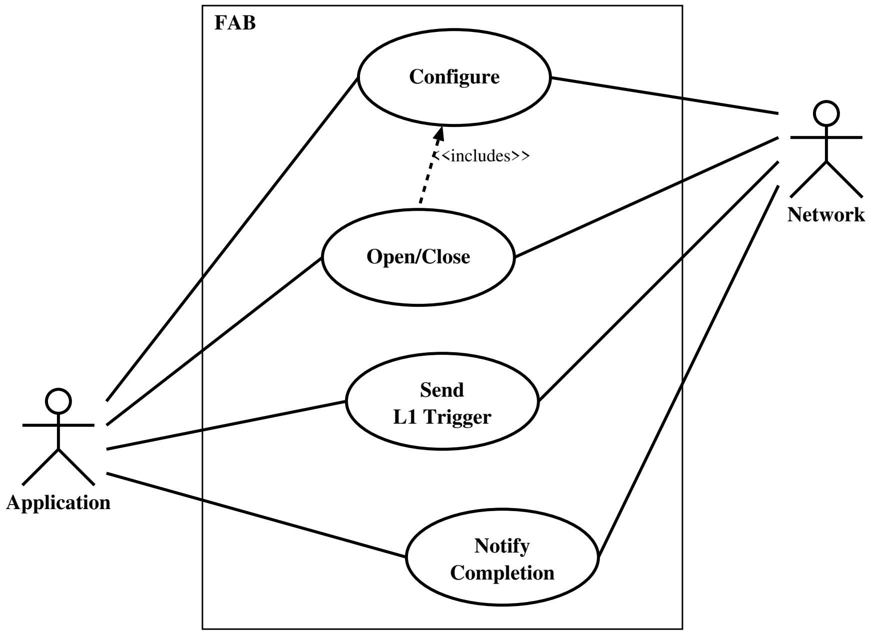

Figure 1: Use case diagram of the system

> The library provides a reliable broadcast on UDP or IEEE1394 network. The library operations are invoked by an application software on the root node of the network and it drives lower level protocol specific libraries. It has its own thread of control to enable asynchronous communication with the application. Operations include placing send request and notification of errors when it occurs, together with the object instanciation/deletion and configuring internal parameters. The API is independent from the choise of underlying network protocol. Negligible execution overhead is preferable.

The use case diagram of the system, derived from above description, is shown in Figure 1. The actor 'Application' is the Readout Manager (RM) software. The actor 'Network' is a kernel driver of either UDP/IP on Ethernet or IEEE1394. The application opens or closes the system. The 'Open' use case includes proper configuration of a driver. It also starts own thread of control. The application place a request to send L1 trigger information. Then, if necessary, it can block waiting for the result, which is reported later by the system. Status information is accessible as well as traffic statistics.

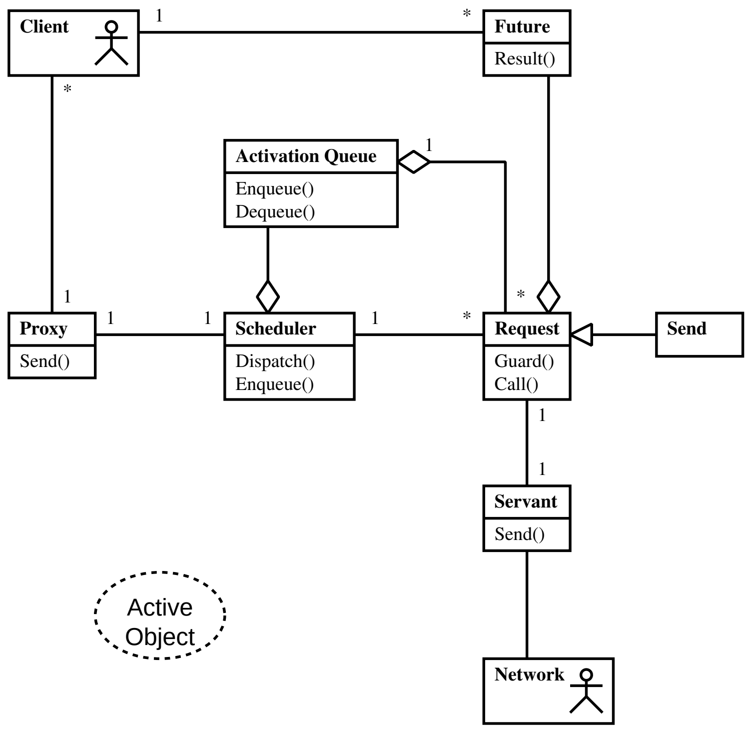

Figure 2: The class diagram of the active object pattern

Because the library has to perform various I/O operation independently from the caller's thread of control, active object pattern is applicable. The class diagram of an active object pattern is shown in Figure 2. The class 'Proxy' is the only one access point for the client. Whole task of the library is devided into two parts, the class 'Scheduler' controlling when requests are executed and the class 'Servant' knowing how to behave on a request. A Scheduler has its own thread of control, which controls the consumption of the requests. The class 'Activation Queue' keeps all requests until these are dequeue by a Scheduler and processed by a Servant. All methods in a Servant have corresponding objects, which are subclass of the 'Request' class. If a method of a Servant is two-way, it returns a object of class 'Future' to a client application. These behavier is also illustrated in a sequence diagram, Figure 3, for the 'Send' use-case.

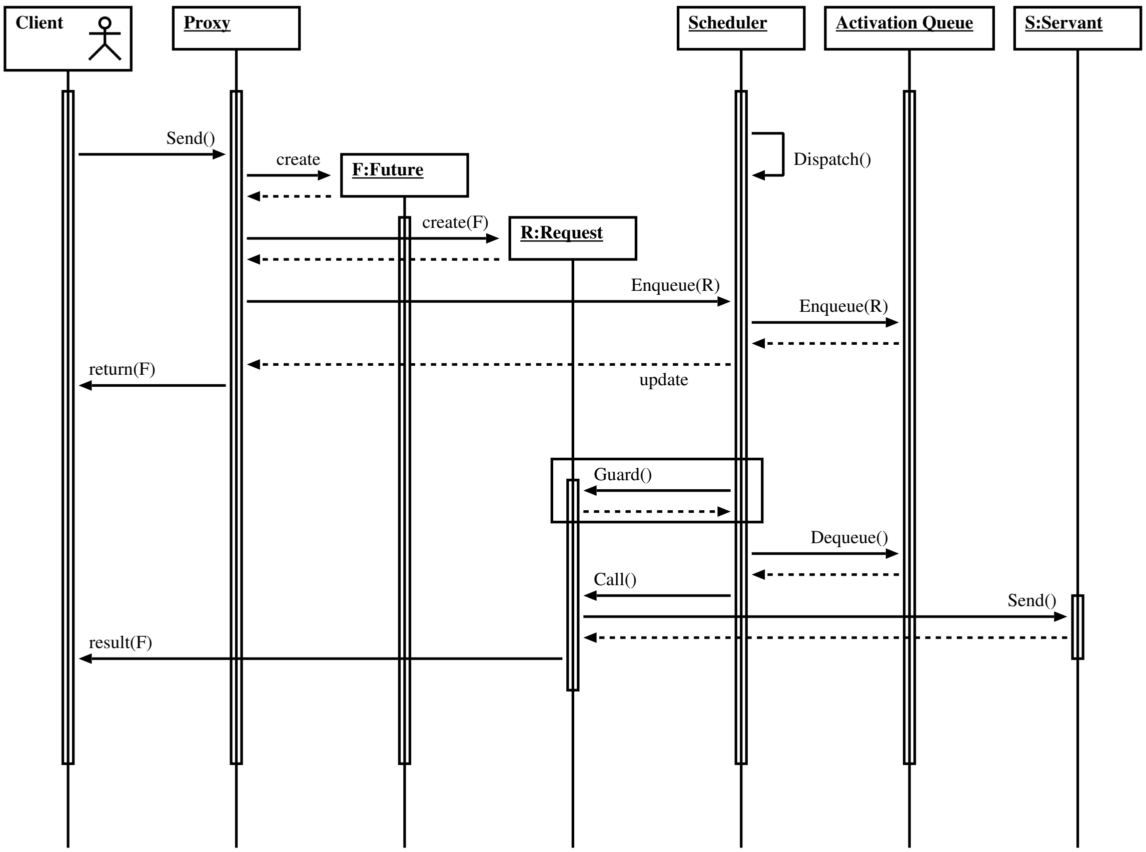

Figure 3: The sequence diagram of the active object pattern sending a packet to the network

3 Architectural class design

We extend the active object classes for the RCN to cope with several possible senarios.

1. A packet of data is sent out to the network using 3-4 FEC. The network is either Ethernet or IEEE1394.
2. A packet of data is sent out but the system received a packet recovery request some time later. Therefore, the system send the packet again.
3. The system is configured so that it waits for at least one ACK from receivers before sending the next data.

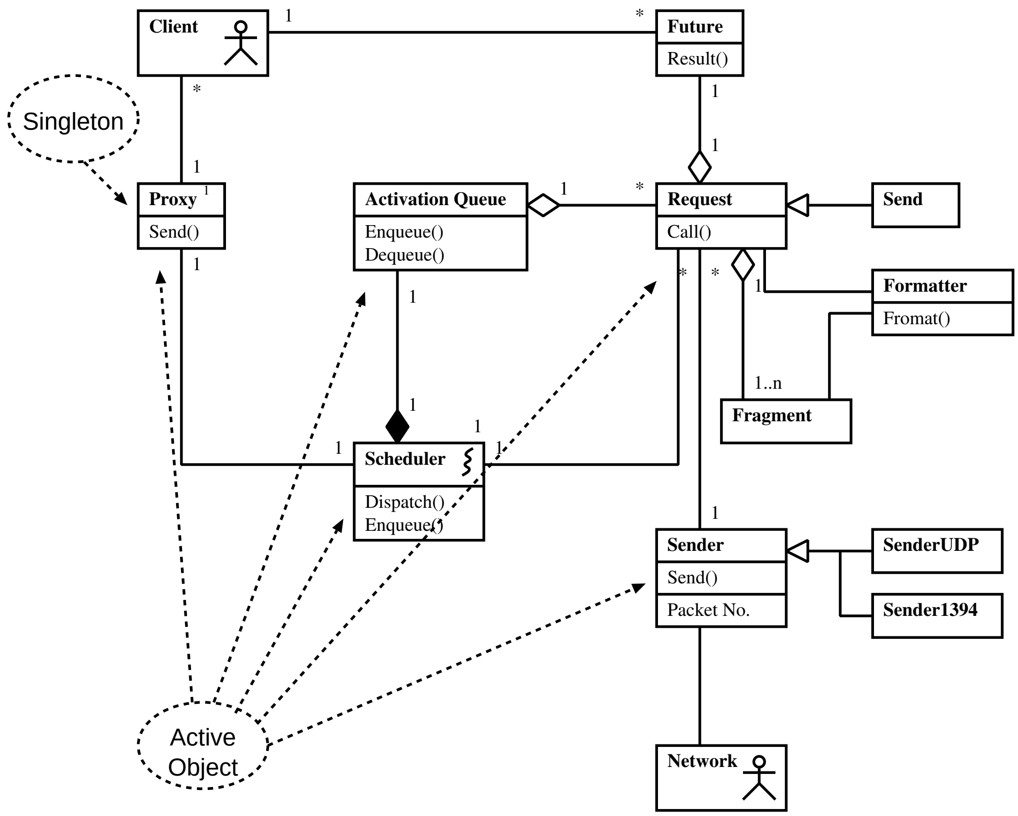

Figure 4: the active object pattern is employed to the RCN application

4. The system is configured to use different FEC scheme or not to use FEC at all.

To accept the first scenario, the active object pattern is used as following. A client, the RM software, ask a Proxy to send a bunch of trigger information. Because the library has to control the network solely, singleton pattarn can be applied to the Proxy. The Proxy creates a Request and pass it to a Scheduler. The Scheduler put it into a Activation Queue. Then, the request is taken out from the queue by the Schduler, fragmented using an Algorithm into a number of Fragments, and sent by a Sender. Every packet sent out by the Sender has unique and sequencial number to detect packet losses at the receiving nodes on the network. We have two kind of Senders, one for Ethernet and the other for IEEE1394.

Four new classes are added as shown in Figure 4. 'Fragment' is a data representation actually sent by the network. 'Formatter' formats original data asked by the client into the Fragments. 'SenderUDP' and 'Sender1394', each for differnet network medium, are subclasses of the Sender, which was called Servant in the active object class diagram. The Sender keeps an attibute 'Packet No.' put on each packet sequencially.

The second senario requires another set of active object classes to receive packets from the network. Receiving a packet repair request from the network and send the packet again to the network is similar to the first scenario beside a thread of control watching the network drives the system instead of the Proxy.

In Figure 5, the Receiver watches out the network to receive a request from nodes, RUs. It has its own thread of control, like the client application. Depending on the network medium, one of two subclasses, ReceiverUDP and Reciver1394 is used. The RUs knows only a Packet No. of the missing packet and the Receiver has to convert it a Request object. 'Active Request Pool' keeps track of Request objects already sent. If corresponding Request is found in the pool, RetryProxy is the conterpart of the Proxy and asks the Scheduler to put the Request in the queue with the highest priority to minimize the latency.

The scenario 3 add another object 'Send Lock' shared with sender side components and receiver side components. This object has to keep a configurable attribute of the operation mode. The last scenario, 4, suggests subclasses of Format. A design pattern, strategy, can be applied here. This design also gives additional flexibility of assigning different Formatter to each Request, for example, control messages can be formatted without FEC while trigger information is done with it.

Next, we visit the Request object lifecycle, Figure 6. In the RCN Protocol, a request stays active until it is delivered to all receivers in the network. Because the protocol is negative acknowledgement (NACK) based, the delivery is confirmed only by a certain time period without any packet recovery request. In the figure, a request is placed in a low priority queue at the first time after creation and appropriate fragmentation by a Formatter. The scheduler picks it up and delegate a send operation to a servant, then the state of the request is changed to 'Active'. If there is no recovery request by receivers for a certain time period, the object is deleted and the exit status 'Success' is propagated to the client application by the Future. The request is queued into a high priority queue when the scheduler received a recovery request due to packet losses. Then, it will be sent later.

How to reap the Request objects and who does it are still issues. In the original active object design using Future object, all requests are checked and deleted if necessary by the client. This method requires significant amount of service at the client side at every data transfer and can not be used for the RCN. A possible solution to avoid explicit object deletion at the client side is applying the 'Asynchronous Completion Token' pattern and

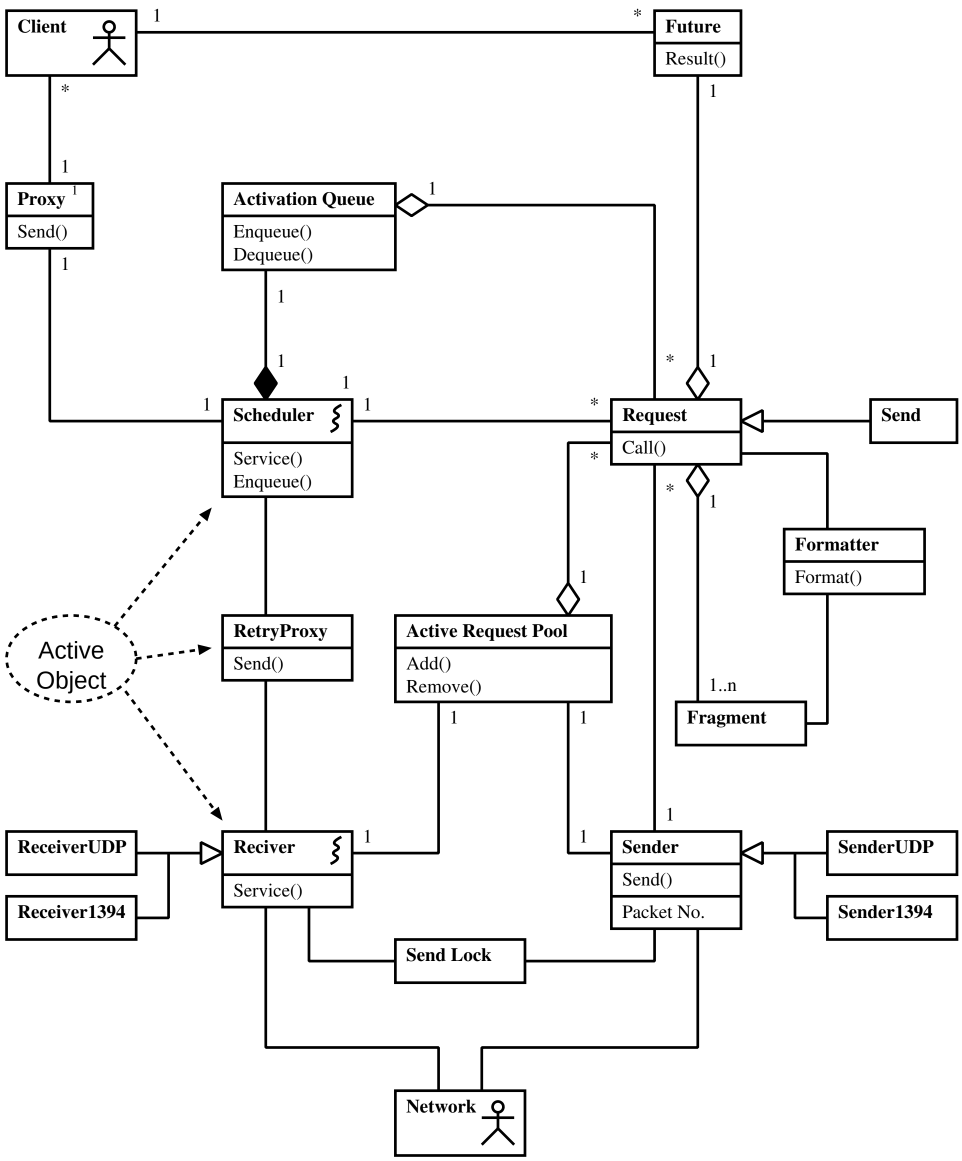

Figure 5: the baseline class diagram capable to send and receive packets to/from the network

let a Completion Handler report to the client only when an error or other special event occurs. In that way, the client can have freedom to treat how to take care of the Request object upon completions, and it become opaque to the library. Objects having its own

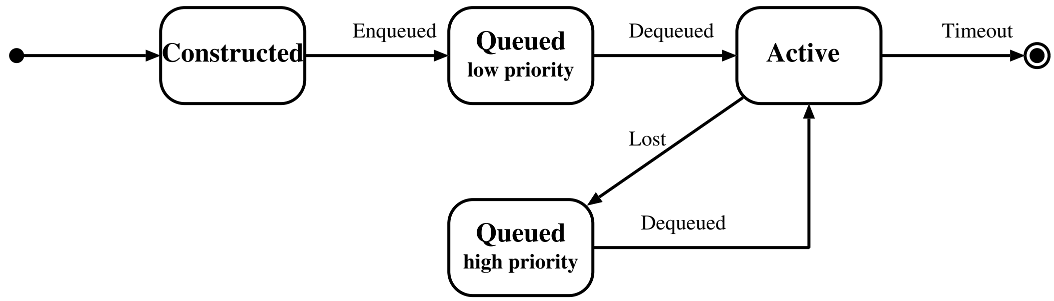

Figure 6: The state chart for a Request object

thread of control, either Scheduler or Receiver are able to invoke the Completion Handler. The Receiver is chosen here, because it is responsible on managing Request objects in the Active Request Pool and determines their state. The resulting design is shown in Figure 7.

Finally, we clean up the class structure. Because only the Send request is discussed so far as a subcleass of the Request, we don't need the Send class. The functionality of the RetryProxy is just to propagate a Request object to the Scheduler. This class, therefore, can be omitted too and integrate the Send() operation into the Service operation of the Receiver. The 'Packet No.' attribute of the Sender is better kept in the Request object, because a Request can be transmitted multiple times but the assigned packet sequence number stays same to track packet losses and recoveries. The associations attached to Active Request Pool is not clear. If the Scheduler can request the Receiver to put a Request into the pool, association between the Sender and the pool is not necessary. Then, Active Request Pool is owned by the Receiver. There are duplicated structures of network specific classes for the Receiver and the Sender. Although abstract factory design pattern can be applied here, the duplicated class hierarchy is too small to introduce such complecated class structure. We add a new class 'Network I/F' which absorbs all the network dependent part. Finally, 'Formatter Factory' class is added to ease switching over various formatters. The original class diagram started from Figure 2 gets to the final structure, Figure 8.

4 Detailed class design

Before implementing the library, we need to clarify associations and class attributes. (The ACK mechanism and the Lock class are not implemented yet.)

Associations are implemented as following.

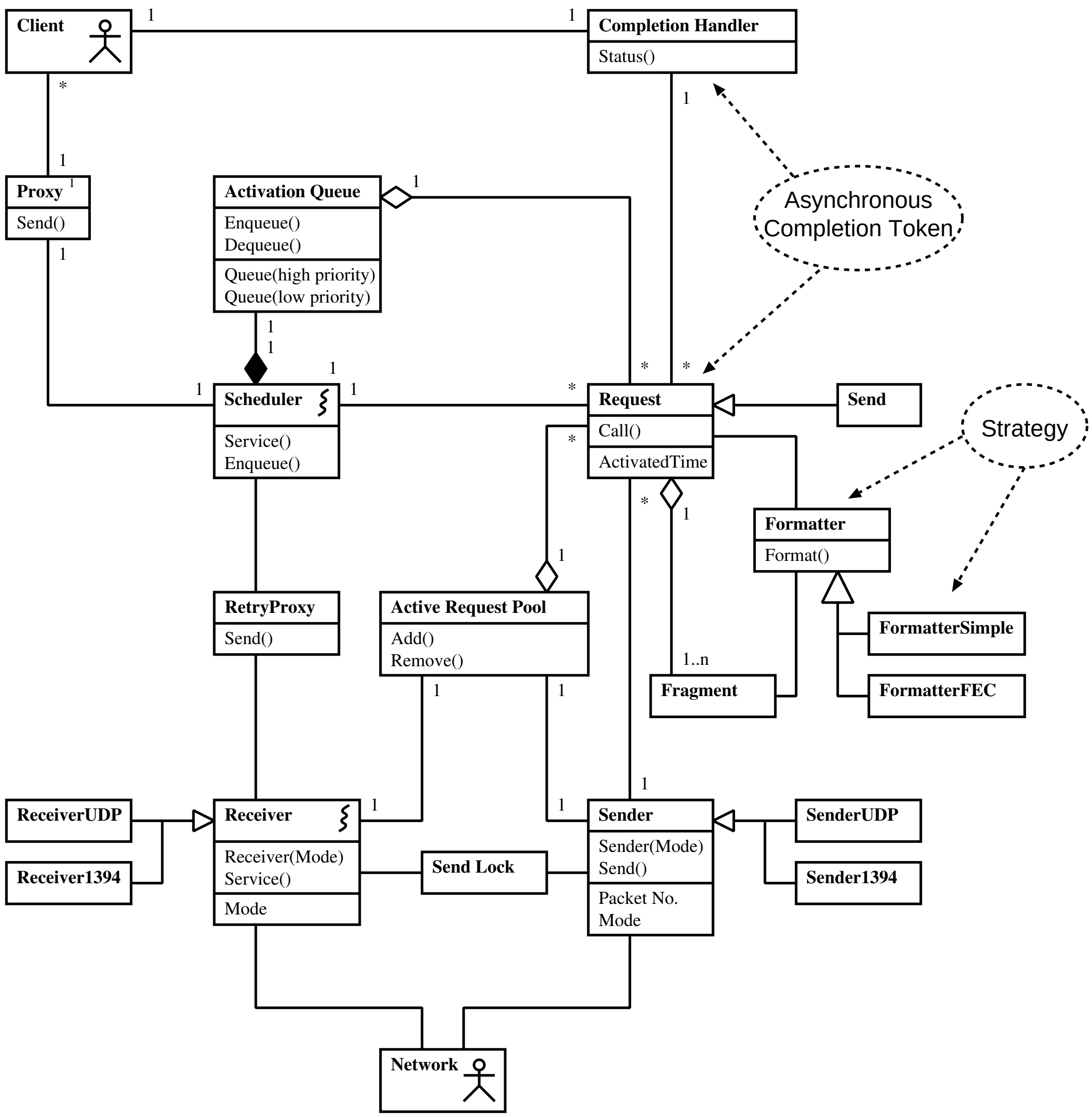

Figure 7: The class diagram taking care of the Request object life cycle

client-Proxy A client calls the Instance() operation of the Proxy class. It returns a reference to the single Proxy object.

client-Completion Handler A client creates its own Completion Handler sub-class'd from the Completion Handler object and place a Send() request with the reference to it.

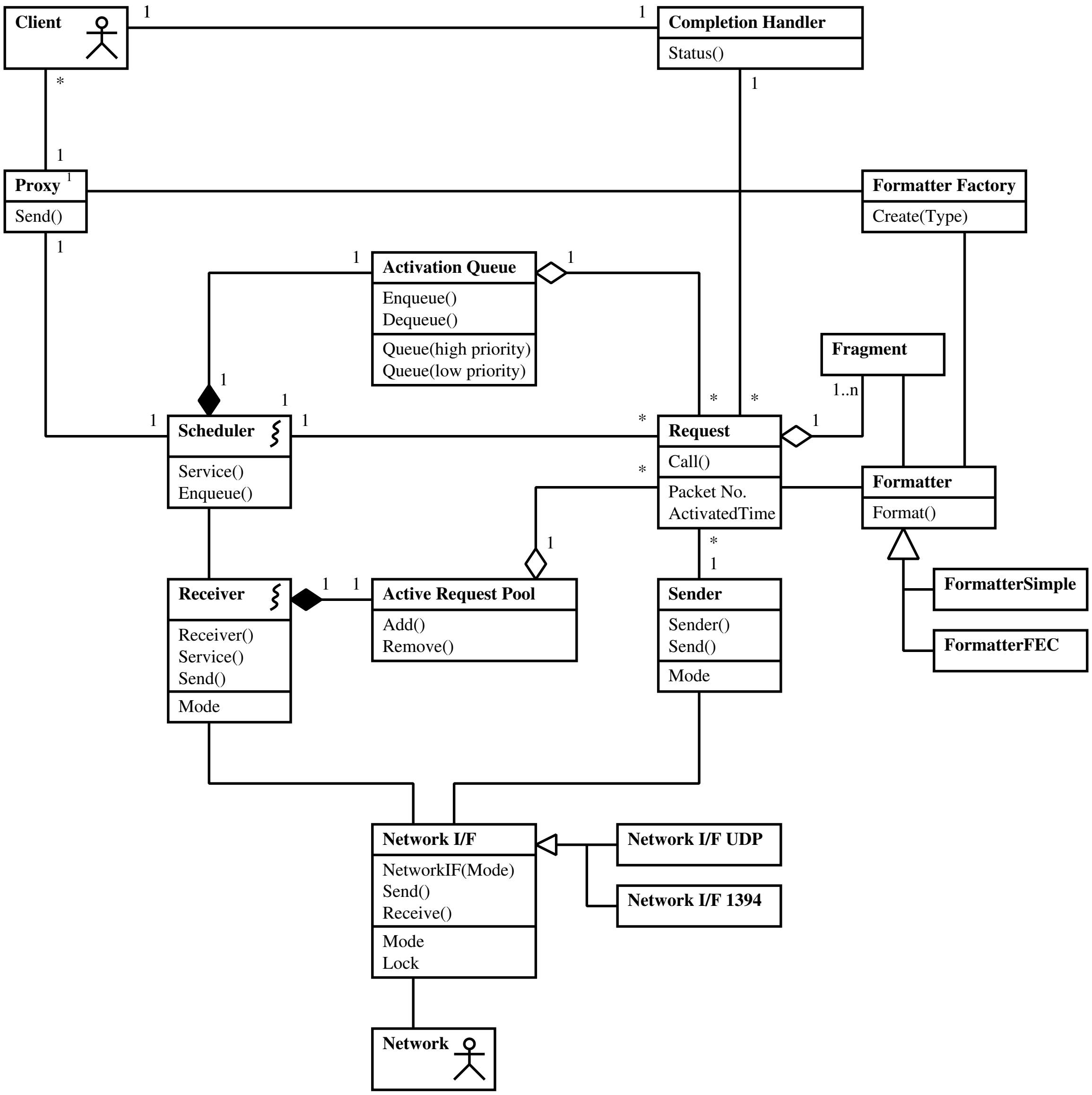

Figure 8: The final class diagram

Proxy-Scheduler A Scheduler object is created in the constructor of Proxy object and the reference of it is kept in the Proxy.

Scheduler-Activation Queue Activation Queue object is an attribute of the Scheduler.

Scheduler-Request A Request object is created by the Proxy and passed to the Scheduler as an argument of Enqueue() operation.

Activation Queue-Request The Activation Queue object receives a reference to a Request by the Scheduler as an argument of Enqueue() operation.

Request-Completion Handler A Request object is given a reference to the Completion Handler object by the Proxy as an argument of the constructor.

Request-Formatter A Request object is given a reference to a Formatter object by the Proxy as an argument of the constructor.

Request-Fragment The Format operation of the Formatter object returns array of references to the Fragment objects, which is an attribute of the Request.

Request-Sender A Request object is given a reference to the Sender object by the Proxy as an argument of the constructor.

Scheduler-Receiver References to the Scheduler and Reciver objects are passed by the Proxy as an argument of the constructors of each class.

Receiver-Active Request Pool Active Request Pool object is an attribute of the Receiver.

Sender-Network I/F A Sender object is given a reference to the Network I/F object by the Proxy as an argument of the constructor.

Receiver-Network I/F A Receiver object is given a reference to the Network I/F object by the Proxy as an argument of the constructor.

Network I/F-Network Operations of the Network I/F object call functions of Network.

The class attributes are shown in Figure 9. The Request class should has a reference to the data to be sent. The Fragment class has actual storage space for the data together with packet number and sequence number in the packet to compose a data stream for the network.

Figure 10 illustrates the packet data format. All fields are a quadred or multiple quadreds. The type-independent part of the header consists of the packet length in unit of quadrets and the packet type. There are two control packet types, Acknowledge and RepairRequest, and several data format packet types indicating how original data is formatted into the packets (just a copy or with FEC etc.).

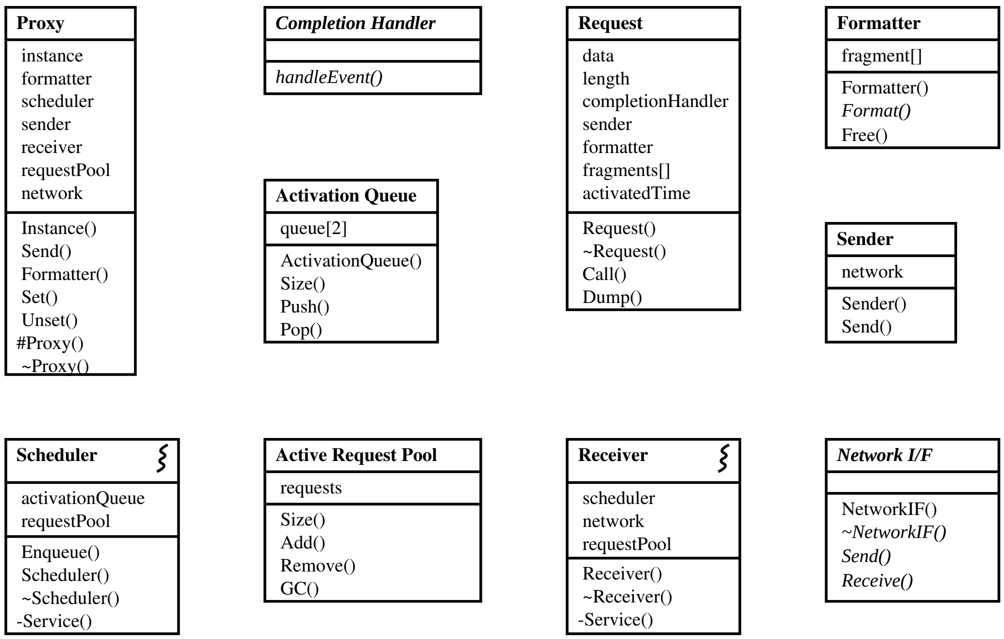

Figure 9: Attributes/operations of each class

For data packets, data ID, sequence number, CRC and data content follow. The data ID is a sequencial and unique number attached to each Request. The sequence number starts from 0 and it is incremented for each following packet generated from the same Request. The receiving node, RU, checks packet loss with these two field. The CRC field is used to check packet corruption. The data contents is an array of trigger information.

Acknowledge packet has a node ID information telling originator and the data ID acknowledged. RepairRequest packet has the node ID, number of requested data IDs and an array of IDs requested.

The threads of control and the synchronization mechanism are implemented using POSIX thread library functions. The objects of Activation Queue and Active Object Pool work as buffers between two threads, therefore, these are protected by a mutex lock. and the threads communicate each other by a condition variable.

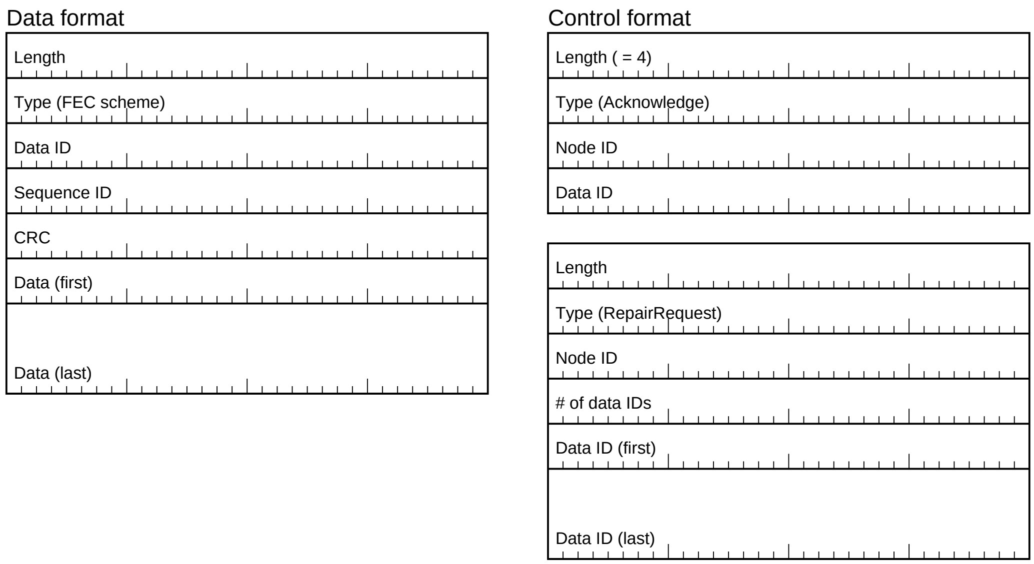

Figure 10: Data format of packets

5 Classes for the leaf nodes

6 Class definition in C++

`http://home.fnal.gov/~ichiro/document/rcnp/includes`

7 Known problems

- A proxy class for the data receiver side (RUs) should be put in the library.
- Definition of packet format is declared in Formatter and many other classes refer it.

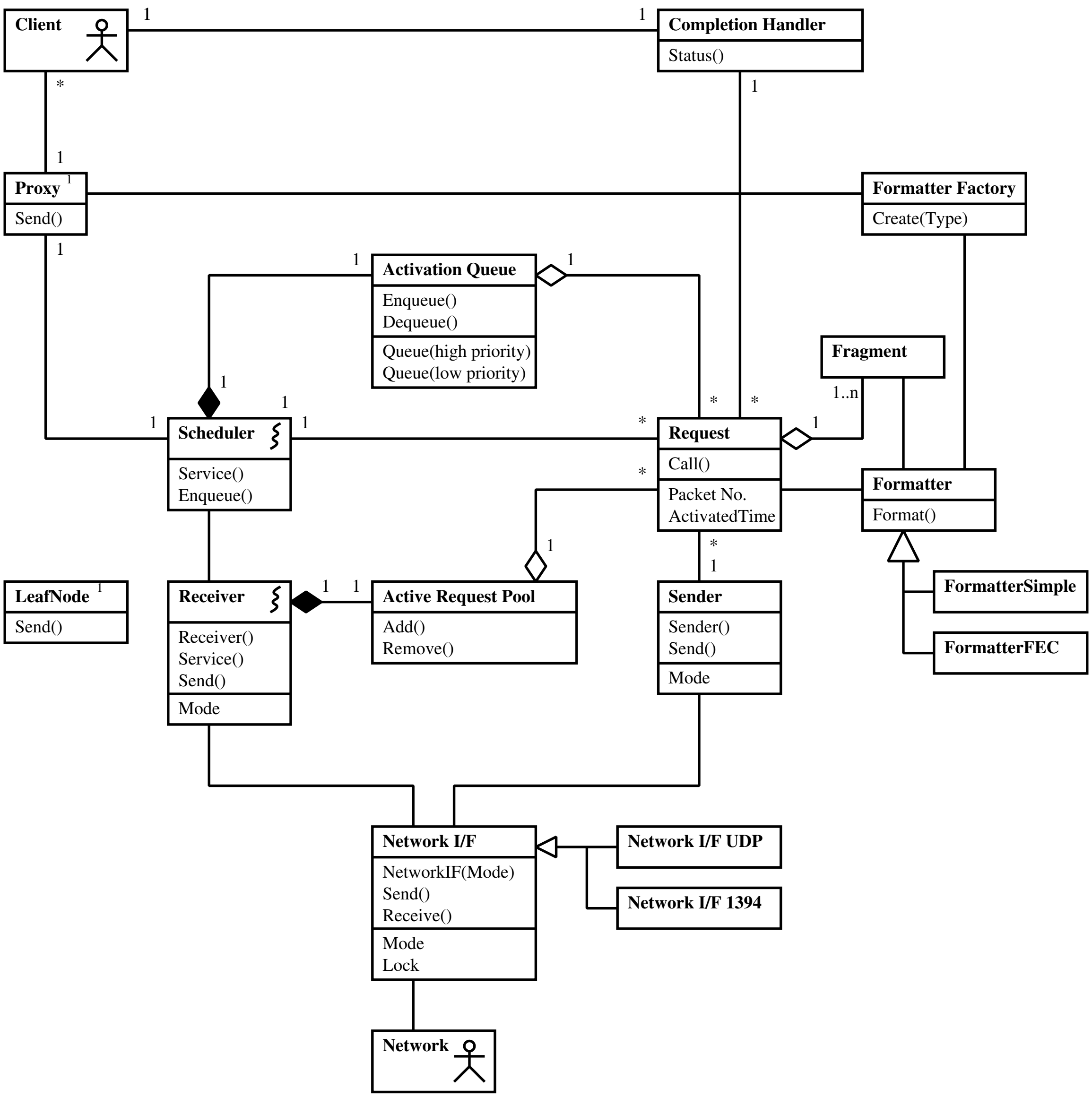

Figure 11: Classes for the leaf nodes