Spring 1997

# On the automatic generation of network protocol simulators

Andrew Chen

*University of Richmond*

Mathematics and
Computer Scienc
Che

# On the Automatic Generation of Network Protocol Simulators

**Andrew Chen**

**Honors Thesis**[1]

**Department of Mathematics and Computer Science**
**University of Richmond**

**April 18, 1997**

1. Under the direction of Dr. Lewis Barnett

This paper is part of the requirements for the honors program in computer science. The signatures below, by the advisor, a departmental reader, and a representative of the departmental honors committee, demonstrate that Andrew Chen has met all the requirements needed to receive honors in computer science.

[illegible]

(advisor)

[illegible]

(reader)

[illegible]

(honors committee representative)

# On The Automatic Generation of Network Protocol Simulators

by

Andrew Chen

## 1. Introduction

Computers communicate with each other over various communication networks via a language known as a protocol. The design of the protocol can have a significant impact on the efficiency (and effectiveness) of the network. Because building an actual network to test the performance (and reliability) of a new protocol is rather expensive and time consuming, there is an interest in simulating network protocols in order to determine how efficient the communication network is. We are therefore interested in automatically generating simulators that could measure the performance of the new protocols.

There are two main parts to this project. The first part is designing and developing a library of general simulation constructs for packet based networks. A packet based network is a network where the information is transmitted in sometimes variable length units called packets. This library should be useful in its own right for the construction of custom simulation programs, but it is primarily intended to aid in the automatic generation of simulators. This library is called the Run-Time Simulation Library (RTSL). The second part is to develop a process whereby a formal specification of a protocol's behavior can be automatically translated into a program that simulates the protocol's performance. Alternatively this could be viewed as an automatic simulator generation tool that generates code which uses the library of general simulation constructs. In general such a tool will be referred to as a Protocol Description Language (PDL) translator.

## 2. Goals

The main goal of this project is to support rapid generation of protocol simulators by automatically translating a high level description of the protocol into actual C++ code that simulates the performance of the protocol. A secondary goal was to produce a standalone library to aid in the creation of most protocol simulators.

In short, one could phrase this as a "Network Protocol Performance Measuring Simulator

Generator". This way of looking at it provides an outline of some of the issues that needed to be addressed, namely networks, protocols, performance, measurement, simulation, and generation.

We would like to specify the protocol via a protocol description language. The automatic simulation generation tool(s) would use the protocol description to create the simulator for a network that used the described protocol. Thus protocol specific issues such as collision resolution[1] and the use or not of a token[2] would be a concern. The goal of these generated simulators would be to determine how well the protocol performs under various loads and traffic patterns.

The notion of the network contains with it the ideas of structure (in terms of what parts there are), layout (how these parts are connected), and behavior (how these parts functionally relate). These will be covered in more depth in a later section.

Since the whole point of the generated simulators would be to measure performance, timing issues (as relating to actually simulating the packet transmissions and collision times) are central to their operation. Since we want to be able to measure the performance, we must be able to get information about such things as transmission delays, collision rates and signal degradation.[3] We want to be able to measure the performance of the network (in terms of packets transmitted, number of collisions and so on) under various traffic conditions. By "traffic" we mean how many packets are sent (i.e., the packet traffic), by which station and when. Thus the code in our simulator will make it a point to keep track of such things. The simulator should be able to read in an experiment description file (EDF) that contains information about the various computers on the network and how much information they are trying to send at what times and process that accordingly.

As the generated code is to be for a simulator, much work needed to be put into the library of general simulation constructs for packet based networks. This constituted the bulk of the implementation.

## 3. Design Issues

There were a number of design issues that were dealt with during the completion of this work. Various event type management concerns severely affected the design methodology. Being a large-scale project, there were some implementation ideas that are generally believed to be very useful for large-scale projects but that I had never seen used. In part this project became an attempt to learn and use these implementation ideas efficiently and effectively. One of these implementation ideas was effective use of the Standard Template Library (STL) (see section 3.2.1). Another one of these ideas was effective use of multiple inheritance. Finally, the powerful operator overloading provided by the Standard Template Library (STL) presented the possibility that the high level protocol design could be expressed as subclasses of various classes in the RTSL rather than as input to the PDL translator. Subclassing is a C++ language feature (it is also present in one way or another in all object-oriented languages) that enables one to inherit whatever is present in certain user designed data types called classes into another and different class. These design issues influenced this work greatly.

### 3.1 Variable Event Type Management Concerns

This simulator framework is based on the concept of events and event-based simulation. In event-based simulation, each change of state in what we are simulating is thought of as an event. Events occur in specific orders, and tend to give rise to other events. The central idiom of dealing with the order of events and processing them correctly is a queue of some sort. In this simulator framework a priority queue based on the event's relative simulated time is used.

#### 3.1.1 Event Field Considerations

There were a variety of concerns relating to the management of the variable number of different event types. One specific concern was that some events need to have certain data associated with them, whereas others do not, or need different data associated with them. This is a concern because we'd like to be able to queue all the events so that we can handle them one at a time (potentially in an order different than how they were created), while still being able to treat different events differently and access associated data only in appropriate contexts. For example, when an event is removed from the queue, it needs to be "downcast" (see below) to its appropriate event subtype in order for event subtype specific members to be accessible.

### 3.1.2 Static versus Dynamic Event Type Management

There were a variety of approaches to dealing with this problem that were considered. Most of the time these approaches involved type casting of one form or another. When unions are used, normal casts are used via the union mechanism. When subclasses are used, the pointer to the parent class needs to be cast down to the level of the child class. This is referred to as downcasting.

#### 3.1.2.1 "Safe" Downcasting

One approach we took to dealing with this problem was "safe" downcasting. In "safe" downcasting, in the abstract[4] parent class there are virtual void methods with the name of "as_<subclass_name>" that in appropriate subclasses either return a null pointer if they are not of that subclass or "this" if they are. This method can be used to determine which subclass an object belongs to after it has been cast to it's parent class. As such, this mechanism is a form of run-time type identification. This sort of static enforcement of the downcasting process only requires the programmer to declare additional appropriate pointers as well as to check to make sure the returned pointer is not null. This declaration of additional pointers is so that whenever referring to an object that uses this feature, in addition to needing a pointer of the type of the parent, a pointer that corresponds to the child class is necessary for the return value of the "as_<subclass_name>" method. While this creates the additional work of having to declare and implement all these methods, it literally forces programmers to not make the sort of errors that can result from things like type codes and a union. The following example[5] should illustrate this:

```
class System_Event;
class Placeholder_Event;

class Simulation_Event : public Ref_Cnt_Obj, public TimeStamp_Obj {
  public:
        Simulation_Event(); // the constructor

        virtual System_Event* as_System_Event() = 0;
                     // more elegant than plain downcasting
                     // returns a null pointer if it is not
                     // a system event
        virtual Placeholder_Event* as_Placeholder_Event() = 0;
                        // more elegant than plain downcasting
                        // returns a null pointer if it is not
                        // a placeholder event
};

class System_Event :    public Simulation_Event
{

//
//    Various data members
//    (and their appropriate data hiding levels)
//    go here
//

  public:

        System_Event()
        {
//
// this constructor a dummy constructor to illustrate the concept
// this constructor not present in real implementation
//
        };
//
//    Various data member specific constructors go here
//

//
//    Various accessor functions go here
//
        System_Event* as_System_Event()
        {
                return this;
        };

        Placeholder_Event* as_Placeholder_Event()
        {
                return 0;
        };
};

// And so code to actual do the downcast would look something like
/*
    System_Event * bob;
    Simulation_Event * fred;

// and now for the creation and upcast

    bob = new System_Event;
    fred = bob;

// and now for the safe downcast
// (This is a trivial example,
// but it could be used if fred was an object placed in a queue.)

      assert(bob = fred->as_System_Event());
*/
```

It should be noted that "safe" downcasting is actually a crude form of run-time type information, and that the use of efficiently implemented run-time type information (or RTTI, as the C++ version of this is called) should be considered as a potential option instead, if the given C++ compiler supports it.

One possible way around the fact that all subclasses need to be known and that each includes the "as_<...>" method implementations is the following: the parent class have implementations of all the "as_<...>" methods and return a null pointer in all of them. This decreases the amount of work needed to add in a new subclass.

#### 3.1.2.2 Event Codes and Unions

Another approach was the use of event codes as a field to identify what the event was. Event codes are typically used in conjunction with unions, but unions were not used here. Instead we didn't share the same memory space within the event object for the various different data that might be stored. The advantage of this was that prior to accessing appropriate fields, liberal uses of the "assert" macro could be used to check to make sure that they were valid. This checking serves to help prevent the sorts of errors typically involved in the type code and union approach. While the "assert" macro can also be used in the type code and union approach, the possibility that an improper member of the union was initiallized can not be checked with the "assert" macro,. However if the memory spaces for the various different data that might be stored are not shared, the use of the "assert" macro helps verify that the proper data type is being used and that improper data types for that event code are null.

#### 3.1.2.3 Type Management Final Decision

A combination of the two above techniques were used. In the current implementation, however, use of "safe" downcasting is not necessary because there is only one type of subclass of the abstract parent event class. A second subclass type[6] was planned but would only be necessary in a much more ambitious implementation.

The one subclass type of the abstract parent event class is called System_Event and uses event codes specific to “system events”. Alternatively we could have not used event codes at all, but this would have required writing additional classes which was not easy during the development process. For all practical purposes we essentially only used event codes here, however with the abstract parent class in place, the support for creating additional subclasses is much easier to provide.

### 3.1.3 Event Life, Death, and Rebirth

The life cycle of the typical event in this simulator framework begins with an event sender creating an appropriate instance of a subclass of the event class. By using the appropriate constructor the relevant fields in the new instance can be set appropriately. The event instance is then placed on the priority queue (priority by time stamp). At an appropriate time, the event is removed from the priority queue. The removal is to facilitate processing by an instance of a subclass of the event receiver class. The subclass takes the event and interprets any appropriate associated data and takes any appropriate action associated with the occurrence of the event. Presently the event queue is only used part of the time, and sometimes the events are sent directly from the event sender to the event receiver. It is anticipated that this “feature” of direct event transmission (by bypassing the queue) will go away.

The queue is a priority queue that uses the ordering conferred by the inherited behavior of the TimeStamp_Obj class to order the events via their timestamps.

## 3.2 Effective use of the STL

In C++ there exists the handy language feature of operator overloading. This can be used to specify how various operators should work on programmer defined types. Also in C++ there exists the language feature of templates, which supports generic programming. Generic programming is the specification of the algorithm irrespective of the type it is being applied to. This enables one to write code that is irrespective of the intended type, so long as the given operators and functions that deal with such types exist for the type. The Standard Template Library, or STL [Musser96], is a collection of routines using these language features to provide much of the common functionality of various abstract data types such as lists, queues, stacks, dequeues, vectors, mappings, and sets.

One of the implementation ideas that are generally believed to be useful is effective use of the Standard Template Library (STL). This has been shown to be useful in this project via the time saved not having to implement most of the event queue and most of the Data_Obj class.

## 3.3 Effective use of multiple inheritance

Another of these ideas was an approach to class library design involving non-trivial productive use of multiple inheritance. In this approach to class library design, the various possible aspects that one might want a particular class to have are separated into parent classes for use in the particular class that is being presently written and for reuse in classes that may be written in the future. For example, in this class library there is a class called "TimeStamp_Obj" which is a class of objects that have a timestamp and are ordered by their timestamp, and a class called "Ref_Cnt_Obj", which is a class which has a reference count and methods supporting helping maintain the reference count appropriately. Thus if we wanted an object with a timestamp and we wanted to keep track of it with reference counts, we would merely construct a subclass of both "TimeStamp_Obj" and "Ref_Cnt_Obj". This technique has been useful in this project in the amount of code that is reused. I claim that it has made the code more compact, modular, maintainable, and has saved development time.

## 3.4 Choosing subclassing over separate language translation

### 3.4.1 A description of the plans for the PDL translator and desirable attributes in proposed separate language

The PDL translator was originally designed with ambitious plans in mind. Specifically, it was intended to translate from a separate language that supported C++ style expression evaluation and multiple inheritance in a goal-directed context(see section 6.4.3.1) into C++.

There were several desirable attributes in the original proposed separate PDL. One of these desired attributes was the ability to check the contents of the packet at the byte level because some protocol actions require this ability (for example, a router or a hub might need to look at header information in a packet before sending it on the appropriate line). Another of these attributes was the ability to check the present state of the line (such as busy or idle). The ability to refer to the present state of the station and/or the node (such as in the process of receiving a packet, sending a packet, or idle) was also desired. Another desired attribute was the ability to use the information available through all these checks to determine what action to take in terms of transmitting a packet, receiving a packet, canceling a transmission, delaying a transmission, changing internal state, or changing the contents of the packet. Conciseness and relatively familiar syntax were also things that were considered desirable attributes. The C++ style expressions have byte-level checking support and are well suited for management state information and indicating appropriate actions at appropriate junctures. C++ style expressions also aid in conciseness and familiar syntax. Multiple inheritance aids in code reusability and therefore in conciseness. A goal-directed context can aid in conciseness.

### 3.4.2 What we lose by choosing subclassing over having a translator for a separate language

A full grammar and lex and yacc specifications for a proposed PDL were developed. These were never incorporated into a working translator because we found that the power of operator overloading enabled most of the features of this proposed PDL to be represented directly in C++. By choosing to use subclassing instead of having a translator, we lose some of the features that we were hoping for, specifically the level of conciseness that would have been derived in part through object-oriented goal-directed techniques (see section 6.4.3.1). C++ lacks goal-directed evaluation, and implementing such a thing in C++ via operator overloading would be a formidable task. We also lose a layer of abstraction that prevents us from really needing to know much about the underlying class structure.

### 3.4.3 What we gain by choosing subclassing over having a translator for a separate language

What we gain is freedom from having to write a translator, faster development time (we need to neither write nor use the translator), and more powerful potential protocol descriptions (i.e., they have access to anything that any normal piece of C++ code does, so they could access the file system, open sockets to receive directions from a client, or even use part of an existing network as part of the simulation).

### 3.4.4 The decision

Given the utility of the operator overloading provided in the Standard Template Library, it was concluded that the bulk of the PDL translation could be trivial, with the non-trivial aspects being parts that were there to deal with non-essential language features of the proposed PDL such as various goal-directed flow of control techniques (see section 6.4.3.1). As a result, the PDL translator[7] was scaled back to having the user of such a simulator write an appropriate subclass and link it with the RTSL.

## 3.5 EDF structural issues

The experiment description file (EDF) is used by the generated simulators to determine the setup of the various computers on the network, how much information the computers are trying to send at what times, and how the simulator should process the measured data accordingly.

### 3.5.1 EDF yesterday

The EDF of Barnett's NetSim[Barnett92] and DQDBsim[Barnett95] packages was structured as a way of specifying where the stations on a given segment of cable would be, how long the cable would be, and various traffic patterns in terms of how much would be transmitted from which nodes at what times and so on.

### 3.5.2 EDF today

The EDF of our framework exists in two forms.

One of these is implemented and is a text file that contains a series of numbers. The first number is the number of events represented in the file, and the remaining numbers are the event codes for the various events. This is primarily intended for testing purposes.

The other EDF of today is called the revised EDF and resembles the EDF of NetSim and DQDBsim packages, but with various syntactic differences that happen to (presently, unfortunately) decrease readability somewhat while increasing conciseness and modularity.

### 3.5.3 EDF tomorrow - NDL?

Originally the revised EDF was intended as a gateway to the NDL[8], or Network Description Language. The NDL was intended to be able to describe the network layout and structure at a higher level than the EDF. The EDF was anticipated to actually specify all the stations and nodes and how they were connected to each other. It was hoped that the NDL would be able to specify the necessary information for a reliable simulation, but not at such a detailed level.

Assuming the problems found in designing the NDL can be overcome, that may eventually replace the EDF. Otherwise, a completed version of the presently revised EDF would probably evolve into a more syntactically pleasing form and then rest there. For example, the following are some lines from the present form of the revised EDF and a hypothetical conception of what the NDL might be:

```
#Revised EDF
*10Base-T copper_wire new thin
+station1 Default_Station 10Base-T 0m
+station2 Default_Station 10Base-T 12m
+station3 Default_Station 10Base-T 24m
+station4 Default_Station 10Base-T 36m
+station5 Default_Station 10Base-T 48m

#Hypothetical NDL
#should do same as above Revised EDF
new media 10Base-T is thin copper wire
create 5 new stations of type Default_Station on 10Base-T with names
"station<station_number>" at 12 meter distances.
```

### 3.6 For future consideration in the design

#### 3.6.1 Virtual machines for simulated behavior

An actual virtual machine for the stations, and nodes, to simulate their behavior would, of course, require a compiler with the virtual machine being the target platform. The advantage of such a virtual machine would be that it may very well ease implementation of certain design issues such as support for protocol layering as well as aid in support of multiple processes at the same station (a long file transfer and web-browsing at the same time, for example).

## 4. Network setup and representation

### 4.1 Network layout

Most network layouts are tree-like in structure, including stars. Some were rings, and some were even doubled-up rings (i.e., biconnected). The NDL was intended to be a higher level representation of network layout than the EDF, but a suitable design for the NDL is yet to be forthcoming. The idea was that we know some information about the network, but want to be able to specify additional details if we feel like it. One example might be a ring based network with a fixed number of nodes on it. After specifying that much information, we already know a substantial amount of information about the network topology. The design difficulty behind the NDL was how to concisely specify the information about a network that is not found in a general description like "ring based network with five nodes on it".

### 4.2 Open-Ear Decomposition, Biconnectivity, and FDDI-1

Open-ear decomposition is a way of decomposing a graph. It seems to be useful (at least in part) in determining if a graph has a two-to-one mapping onto it. At the very least, it is appropriate to finding a way to efficiently traverse a graph in parallel, and as such has connections to networks, as network layouts can be represented in graphs.

Biconnectivity, or the existence of a two-to-one mapping onto a graph, can be used to ensure reliable message synchronization. This biconnectivity in FDDI-1 (a packet-based network protocol designed for a ring topology) is a way of achieving a high degree of reliability. It does this via using the an alternate possible routing through it in the event that it detects that a line adjacent to it is down. FDDI-1 is an example ring based network that we would like to be able to specify in the NDL. This protocol could potentially be simulated via our framework.

There may be an as of yet unexplored connection between open-ear decomposition and possible NDL design.

## 5. Class Hierarchy Details and History

The class hierarchy for the RTSL and complete source code appear in Appendix A. The class hierarchy changed throughout the design process as a greater understanding of some advantageous ways of using the STL and multiple inheritance were learned. For different class hierarchy diagrams as represented throughout the design process, see Appendix B.

### 5.1 Events

There are five classes that deal with events.

#### 5.1.1 Simulation_Event

This abstract class is the parent of all events in this simulator. All events should override the virtual void methods of Simulation_Event. The virtual void methods that begin with "as_" should be used for the "safe" downcasting mentioned in section 2.2.2.1. Likewise, this class (and perhaps its subclasses) need(s) to be edited to support a new "as_" virtual method anytime any new subclasses of this class are written. This does seem at times to defeat the "library" concept. This could be avoided by use of RTTI, or this could be worked around by having a subclass of the class that the "library" knows about, but that the library user implements which would contain all the appropriate "as_<...>" methods and so on.

### 5.1.2 System_Event

Presently this is the only subclass of the Simulation_Event class. This class can be used directly and need not be subclassed. This class has a rather fat interface (the many constructors and access functions in this class frequently have arguments that would be more appropriate for more specific subclasses of Simulation_Event) and presently is a catch-all event type primarily used during the development phase of this framework. Something to consider is potentially breaking this class up into different subclasses of Simulation_Event. By breaking this class up we would gain more memory efficiency (each event instance would only have memory allocated to it that corresponded to what it needed, as opposed to now, when each event instance has memory allocated to it for everything it could possibly need) as well as compile-time type checking as opposed to the run-time type checking provided via the liberal uses of the assert macros presently being used.

### 5.1.3 Event_Receiver

All classes that receive events should be a subclass of this abstract class, and should override the appropriate virtual void method, which is "Handle_Event(Simulation_Event *)". Presently most subclasses of this class will implement the overridden method with a safe downcast to "System_Event *" and then dispatches to various other methods declared and defined in the subclass. However, for any other subclass type that they should be able to handle they should process the appropriate safe downcast to that class. Presently these other various methods that are dispatched to correspond to the various appropriate event codes.

### 5.1.4 Event_Sender

All classes that send events should be a subclass of this abstract class. When we want to refer to a class that sends an event we can just refer to it as an "Event_Sender *" and not via what it actually is. Most of the methods provided by this class are to associate a string with a given "Event Receiver *" instead of having to refer to it directly. This functionality is anticipated to be useful in conjunction with the revised EDF as the revised EDF is anticipated to refer to specific nodes and stations via a name which is most easily represented as a string.

#### 5.1.5 Event_Queue

This is a subclass of a templated instantiation of a priority queue class of Simulation_Event pointers. The priority queue class came from the STL.

### 5.2 Simulation_Agent

This class is for managing the entire simulation process. In a sense, one could refer to this as the class that embodies the "big picture" of what's going on in the simulation. This is the class that reads in the EDF and sets up the event queue initially based on that. This is also the class that manages the event queue.

This class contains a lot of utility functions to manage the simulator overall, such as ReadEDF, Do_Simulation, and various others.

#### 5.2.1 Queue Management

The event queue is a data member of the Simulation_Agent class, and all accesses of the event queue are through the Simulation_Agent class. This makes changing the queue interface easy, as only this class deals with the queue.

#### 5.2.2 ReadEDF

This is the method that reads in the EDF and sets up the queue appropriately. Sometimes it is convenient to override this method so that the EDF is not read, but certain other events are put directly on the event queue. This is useful for testing event types for which the input convention has not yet been defined or implemented. A new version of ReadEDF[9] is needed for the revised EDF to be implemented.

#### 5.2.3 Do_Simulation

This pops an event off of the event queue, processes the event, makes sure the processing of the event was successful, checks to make sure the event queue isn't empty, and repeats. If any of the checks fail, it exits out of the loop and returns to its caller.

#### 5.2.4 Name2Node

This method of Simulator_Agent would serve as a mechanism for keeping track of the nodes via a name (a "char *"). This would aid the workings of the revised EDF.

### 5.3 Timing

#### 5.3.1 Time_Class

This is an extensible, encapsulated class for dealing with time. Its interface is presently rudimentary. The central focus of its interface is the support for the operators "<" and ">" for comparison of two Time_Class instances to determine which represents an earlier or later time. The time class presently is only accurate to seconds. Accuracy to billionths of a second would be useful[10].

#### 5.3.2 TimeStamp_Obj

This class has an instance of the Time_Class as a data member. This class also has the "<" and ">" operators overloaded to call the appropriate operators in the instance of the Time_Class object. TimeStamp_Obj is different than Time_Class because the (public) subclassing mechanism is intended to represent "is a" relationships, and so subclasses of TimeStamp_Obj "are" TimeStamp_Obj, whereas a public subclass of Time_Class would "be" a Time_Class, and we wouldn't want events to "be" Time_Class, as events aren't time, they merely have a time associated. This problem with meaning in subclassing could be remedied with private or protected inheritance instead of public inheritance, but that would render the overloaded operators inaccessible to functions outside of the scope of the (private or protected) subclass of Time_Class. This would not be desirable because the whole point of providing these overloaded operators was so they could be used by functions outside of the scope of subclasses to order the elements of the classes. Thus we have the following two classes: Time_Class and TimeStamp_Obj.

### 5.4 Logger

In essence, this is a library that one can inherit which is designed for use reporting information such as errors. The only data member is a reference to the "ostream" to be used to report the information to.

```
        Logger foo(cerr);
        foo.log("Error","some error");
```

would write "Error: some error" to cerr. The logger class is intended to aid in the generation of trace information about protocols that may not have been proven correct[11] so that their flaws, if any, may be diagnosed. The logger class is also suitable for, and intended to aid in, the reporting of performance data.

## 5.5 Station

In the context of the simulators based on this framework, a station is an origin or destination of packets. As a subclass of the Event_Receiver class, The Handle_Event method is implemented in the Station class and calls appropriate virtual void methods for the various events. The Station class is also a subclass of Event_Sender. Typically it will respond to various events by sending various events.

Presently the only subclass is the Default_Station class, which provides minimal implementations of the event handling routines that the Handle_Event method in the Station class dispatches to. Default_Station is also a subclass of the Logger class so that activity noted by any Default_Station will be "logged". Other appropriate subclasses might be subclasses that generate events appropriate to transmissions at appropriate times.

## 5.6 Nodetype

In the context of the simulators based on this framework, a node is a connection to at least one piece of media that neither originates nor serves as the destination for any packet, but instead passes the packet along (processing it in some ways where applicable) when that is what will aid in the packet arriving at its intended destination (as far as the node can tell).

As a subclass of the Event_Receiver class, The Handle_Event method is implemented in the Nodetype class and calls appropriate virtual void methods for the various events. The nodes play the various roles of serving as the connection between the station and the media as well as connecting different sections of media (as in a hub, router, or repeater).

The Nodetype class is also a subclass of Event_Sender. Typically nodes respond to various events by sending various events.

### 5.6.1 Station_Connection Subclass

This abstract class responds appropriately to the various events that might apply to nodes that are connected to stations.

### 5.6.2 Station_Connection_Echo Subclass

This is a subclass of the Station_Connection class and merely sends back to the station any events that it receives.

### 5.6.3 Station2Station_Connection[12] Subclass

This class was created to test the design structure of a prototype. Instances of this class are intended to serve as a gateway between two stations, with no media. At present this class receives events from two stations and always sends them to the same one of them. The one that always receives the events is specified in the use of the class.

## 5.7 Ref_Cnt_Obj

This class is intended to be inherited by any class for which we want to use reference counts to do memory management.

### 5.7.1 Static Methods

There are a number of static methods in this class which are intended to make it easier to manage and manipulate the reference count values. These include methods to increment the reference count, decrement the reference count (and destroy if it becomes less than one)[13] , and safely destroy (checking to make sure the reference count is one or less)[14] .

### 5.7.2 Event Destruction

As events are subclasses of Ref_Cnt_Obj, it was intended that the reference count mechanism be used to determine when to delete the events.[15]

## 5.8 Data_Obj

The Data_Obj class is intended to represent variable amounts of potentially changing data. The Data_Obj class is a child class of a templated instantiation of an STL mapping from unsigned long integers to characters. This means that anyone can refer to a specific character by providing an instance of this class with an unsigned long integer that corresponds to the character that one wants to refer to. The Data_Obj class inherits most of its behavior from its parent, but contains a few methods to facilitate transferring its contents to other instances of the same class or to character arrays. By using the mapping, the amount of memory used to represent this is only dependent on the number of unique characters accessed via the mapping mechanism, and not dependent on some constant fixed upper bound.

## 5.9 Sim_Packet

The Sim_Packet class is a child class of the Data_Obj and Ref_Cnt_Obj classes. The Sim_Packet class is a class designed to simulate packets in this simulator. All of its behavior was inherited from its parents.

## 5.10 Medium[16]

The medium class handles medium properties and interactions such as signal degradation over time as well as how long it takes for a signal to get from one node to another based on their distances.

## 5.11 Control_Region[17]

The control region class handles interactions between the protocol and the medium, translating things like two signals traveling through the same location into a collision and signal degradation into a higher probability of a transmission error (and altered data in the packet). In essence, one could think of the Control_Region class as representing the world of the protocol (which is medium independent and "thinks" in terms of packets) to the world of the medium (which is protocol independent and "thinks" in terms of signals), and vice versa.

## 6. Summary

### 6.1 Accomplishments

What we have is a framework for a simulator, which means we don't have a generator or a simulator because we opted to subclass and haven't completed enough of the framework. The present implementation can handle simple event and packet transmissions from various created stations and nodes to themselves and sometimes others.

Creation of a simple simulator should merely consist of subclassing the appropriate classes to add in protocol specific behavior as well as to establish that those classes should be instantiated. This could then be compiled and linked with the rest of the library of simulation constructs to create a simulator for the network protocol that would read in a description of the experiment to be run and then run an appropriate simulation.

The structure of an event-based simulator poses considerably complexity and challenge. This suggests to me the need for well developed, flexible, portable libraries of routines and classes to deal with event handling.

I found effective use of multiple inheritance in the process of abstracting properties required for simulation into parent classes and then inheriting those desired properties into appropriate subclasses. The Sim_Packet class is an excellent example of this because it inherits its behavior from both the Data_Obj class as well as the Ref_Cnt_Obj. Some other examples are the Station and Nodetype classes, as they are both subclasses of both Event_Sender and Event_Receiver.

The greatest obstacle to not using the STL effectively is not knowing what is in the STL. Most traditional data structures have been implemented in the STL, as have many algorithms as well. The STL was very useful, as it provided the queue in the event queue and the mapping in the Data_Obj.

### 6.2 Completed goals

The timing related classes are well used and robust, if not complete. The Sim_Packet class is complete. What we have so far can be measured well, and reported well via the Logger class. The event management scheme was the bulk of the problem, and seems to be sufficiently robust right now for further development on other sections of this project.

## 6.3 Uncompleted goals

We have no generator right now, nor do we have a simulator yet. No actual network has been created, not even in demo prototypes with hard-coded events. (It's hard to have a network with no medium.) No protocols have been tested because there has been no network to test them on.

## 6.4 Where to go from here

### 6.4.1 Complete remaining classes

When the remaining classes are complete, we should have an appropriate framework for a simulator. Note that the Medium class will still need to be subclassed to provide support for a specific medium, and that a subclass of both the subclassed medium class and the Control_Region class will need to be created in order to provide a suitable interface to the other classes for sending packets as signals through the medium.

### 6.4.2 Complete remaining design issues

There are still several design issue questions that remain to be dealt with such as proper use of the Name2Node method, how to keep track of medium specific information that would be associated with the packet as it travels, as well as what sort of interface the user of such a framework would find suitable to subclass. Also unaddressed, although the structure is in place for it, is the level of description of traffic load as should be described in the EDF.

### 6.4.3 Support for protocol layering?

#### 6.4.3.1 Object-Oriented Goal-Directed Techniques And Their Applicability Here

The idea behind aiming for goal-directed specifications (a la Prolog; see Appendix C for more details) was that the designer of appropriate protocol simulation implementations might be more able to focus on the behavior of the components involved in the implementation of the protocol and less on how they would go about behaving that way.

The idea behind aiming for this to be done in an object-oriented context was that since various protocols share certain common characteristics, it should be possible to easily implement new protocols in the simulator by inheriting into the new protocol the behavior of the most similar existing protocol and then overriding or providing the differences.

#### 6.4.3.2 Implementation of Protocol Layering

The problem of protocol layering is an awkward one at best. Protocols typically exist in layers, with one protocol using the services of a lower-level protocol to implement itself and provide additional functionality. Presently this design is not suited for protocol layering. However, once the issue of how to represent protocol layering is adequately addressed, the interface of that to the rest of this simulator should not be all that difficult to implement, as this simulator contains those classes intended for low-level simulation (media, for example) as well as those classes that are appropriate at any level (stations, for example). Layering would merely provide additional layers of interfaces between additional classes.When this is represented as merely providing additional subclasses, it is not difficult to implement, but requires that the writer of the subclass be aware of much more of the class structure of this simulator than would be presently required if the framework were complete.This would seem to suggest that at this level the use of a PDL translator would be appropriate. In fact, given not only that but the multi-tasking nature of certain protocols in terms of having multiple layers simultaneously layered on top of them (for example, having TCP or UDP on top of PPP), this would seem to suggest at least considering the virtual machine idea found in section 3.6.1

## 6.5 Related Work

There exist similar systems that instead of translating into a program that simulates the protocol's performance, translate into actual implementations of the protocol. Among these systems are LOTOS and Estelle [Hoffman93, Manas88, Oechslin95, Vuong88].

[Barnett92] Barnett, Lewis, *1993 SIGCSE Paper: An Ethernet Performance Simulator for Undergraduate Performance Networking*, Proceedings of the ACMSIGCSE Technical Symposium 1993, pp. 145-150

[Barnett95] Barnett, Lewis, *DQDBsim User's Manual*, University of Richmond Math and Computer Science Department Technical Report TR-95-01 June 1995

[Hoffman93] Hoffman, B. and Effelsberg, W., *Efficient implementation of Estelle specification.* Techincal Report 3/93, Universitat Mannheim, Mannheim, Germany, March 1993

[Manas88] Manas, J. A. and de Miguel, T., *From LOTOS to C*, in *Proceedings of the International Conference on Formal Description Techniques for Distributed Systems and Communications Protocols*, pages 79-84, 1988

[Musser96] Musser, David R., *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*, Addison-Wesley Publishing Company Inc., 1996

[Oechslin95] Oechslin, Philippe, *Implémentation Optimisée de Protocoles a Haut Débit.* PhD thesis, EPFL Lausanne, Lausanne, Switzerland, 1995

[Vuong88] Vuong, Son T., Lau, Allen C., and Chan, R. Issac, *Semiautomatic implementation of protocols using an Estelle-C compiler. IEEE Transactions on Software Engineering,* 14(3):384-393, March 1988

[1] Collision resolution is the method whereby a protocol handles the situation where packets collide and thereby prevent the information in the packets from being reliably determined.
[2] A token is a special signal used to indicate that the receiver of the token may safely transmit information. Typically tokens are used in a ring configuration network.
[3] Collision rates are the rate at which collisions occur. Ideally, we want the collision rates to be very low while still transmitting and receiving as many packets as possible. Signal degradation is the weakening of the signal (and therefore of the ability to get information from the signal) over time and distance. Signal degradation causes problems because packets are transmitted via signals.
[4] An abstract class is a class with some virtual void methods. Virtual void methods are methods that are declared as part of the class but are never defined. The class is called abstract because it never exists in reality. Only subclasses that implement the necessary declared methods can be instantiated.
[5] This example is based on actual code, but was trimmed down to illustrate the concept of "safe" downcasting.
[6] The second subclass type was intended for keeping track of the internal state of the various nodes and stations so that the actual processing of the events that would occur there could occur in as faithful as possible a simulation (i.e., that two nodes might actually be processing different events "at the same time" so that emulated time-slicing between them might occur, with the second subclass type intended to keep track of which "clock cycle" (so to say) the respective nodes and stations were on).
[7] Granted, it's not much of a translator anymore, but still, we refer to it as a PDL translator for historical reasons.
[8] At the time this document was prepared, the NDL was not designed.
[9] This revised version of the ReadEDF method has not been implemented at the time this document was prepared.
[10] The implementation of this feature was incomplete at the time this document was prepared.
[11] There are various techniques that have been developed for proving protocol correctness.
[12] The implementation of this class was not entirely complete at the time this document was prepared.
[13] This aspect of the design had not been fully tested as of the time this document was prepared.
[14] This aspect of the design had not been fully tested as of the time this document was prepared.
[15] This aspect of the design has not yet been fully implemented as of the time this document was prepared.
[16] The implementation of this class was incomplete at the time this document was prepared.
[17] The implementation of this class was incomplete at the time this document was prepared.

# Appendix A: Source Code

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Control_Region.h

*/

#ifndef CONTROL_REGION_H

#define CONTROL_REGION_H

#include "config.h"
#include "Event_Sender.h"
#include "Event_Receiver.h"

class Control_Region :  public Event_Receiver,  // it receives events
                                                public Event_Sender             // and
 it sends events
{
  protected:
        ~Control_Region();      // it's protected because we never want to
                                                // implicitly destruct one of these

  public:
        Control_Region();       // the contructor

        virtual Result_Code* Handle_Event(Simulation_Event* the_Event);

                // all subclasses of Event_Receiver must provide their own version
                // of Handle_Event
};

#endif
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Data_Obj.C
*/

#include "Data_Obj.h"

Data_Obj::Data_Obj(void)
{
// do nothing constructor - nothing to initialize
}

bool Data_Obj::CopyIn(unsigned long offset, Data_Obj* the_data)

        // CopyIn copies the Data_Obj that is the_data to the offset
        // in this Data_Obj
{
        iterator i;
        for (i = the_data->begin(); i != the_data->end(); i++) {
                (*this)[(*i).first + offset] = (*the_data)[(*i).first];
        };
        return true;
}

bool Data_Obj::CopyIn(
        unsigned long offset,
        unsigned long amount,
        char* the_data
        )

        // CopyIn copies the actual memory contents that the_data points to
        // to the the offset in this Data_Obj.
        // Only "amount" characters are copied.
        // Roughly corresponds to memcpy(c std) or BlockMove(MacToolBox)

{
        unsigned long i;
        for (i = 0; i < amount; i++) {
                (*this)[i + offset] = the_data[i];
        };
        return true;
}

bool Data_Obj::CopyOut(
        unsigned long source_offset,
        unsigned long amount,
        Data_Obj* the_destination_area,
        unsigned long destination_offset
        )

        // CopyOut copies amount characters (using default values if necessary
        // for the unaccessed parts of Sparse_Data_Objects) starting at
        // source_offset from this Data_Obj instance to the_destination_area
        // at destination_offset

{
        iterator i;
        for (   i = find(source_offset);
                        (i != end())&&((*i).first < (source_offset+amount));
                        i++)
        {
                (*the_destination_area)[(*i).first + destination_offset] = (*this)[(*i
).first];
        };
        return true;
}

bool Data_Obj::CopyIn(
        unsigned long source_offset,
        unsigned long amount,
        Data_Obj* the_source_data,
        unsigned long destination_offset
        )

        // CopyIn copies amount characters (using default values if necessary
        // for the unaccessed parts of Sparse_Data_Objects) starting at
        // source_offset from the_source_data to this Data_Obj
        // at destination_offset

{
        iterator i;
        for (   i = the_source_data->find(source_offset);
                        (i != end())&&((*i).first < (source_offset+amount));
                        i++)
        {
                (*this)[(*i).first + destination_offset] = (*the_source_data)[(*i).fir
st];
        };
        return true;
}

bool Data_Obj::CopyOut(
        unsigned long source_offset,
        unsigned long amount,
        char* the_destination_area
        )

        // CopyOut copies amount characters (using default values if necessary
        // for the unaccessed parts of Sparse_Data_Objects) starting at
        // source_offset from this Data_Obj to the_destination_area

{
        unsigned long i;
        for (i = 0; i < amount; i++) {
                the_destination_area[i] = (*this)[i + source_offset];
        };
        return true;
}
```

```
/*
	Run-Time Sumulation Library version 0.01pa
	(c) 1996 Andrew Chen

	No guarrantees.

	Data_Obj.h
*/

#ifndef DATA_OBJ_H

#define DATA_OBJ_H

#include <bool.h>
#include <map.h>
#include "config.h"

class Data_Obj : public map< unsigned long , char, less<int> > {

  private:

  protected:

  public:
	Data_Obj(); // default constructor
	virtual bool CopyIn(
		unsigned long offset,
		unsigned long amount,
		char* the_data
		);

		// CopyIn copies the actual memory contents that the_data points to
		// to the the offset in this Data_Obj.
		// Only "amount" characters are copied.
		// Roughly corresponds to memcpy(c std) or BlockMove(MacToolBox)

	virtual bool CopyOut(
		unsigned long source_offset,
		unsigned long amount,
		char* the_destination_area
		);

		// CopyOut copies amount characters (using default values if necessary
		// for the unaccessed parts of Sparse_Data_Objects) starting at
		// source_offset from this Data_Obj to the_destination_area

	virtual bool CopyIn(unsigned long offset, Data_Obj* the_data);

		// CopyIn copies the Data_Obj that is the_data to the offset
		// in this Data_Obj (if the_data is a Sparse_Data_Obj,
		// presumably some sort of default value would get returned,
		// and the object would seem to be of range from teh first
		// access to the last access

	virtual bool CopyOut(
		unsigned long source_offset,
		unsigned long amount,
		Data_Obj* the_destination_area,
		unsigned long destination_offset
		);

		// CopyOut copies amount characters (using default values if necessary
		// for the unaccessed parts of Sparse_Data_Objects) starting at
		// source_offset from this Data_Obj instance to the_destination_area
		// at destination_offset

	virtual bool CopyIn(
		unsigned long source_offset,
		unsigned long amount,
		Data_Obj* the_source_data,
		unsigned long destination_offset
		);

		// CopyIn copies amount characters (using default values if necessary
		// for the unaccessed parts of Sparse_Data_Objects) starting at
		// source_offset from the_source_data to this Data_Obj
		// at destination_offset

};

#endif
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Default_Station.C

        An off-the-shelf working Station subclass.

  To Do:
        I need to make sure that this deals with the reference counts of
        my_connection_node properly.
*/

#include "Globals.h"
#include "System_Event.h"
#include "Default_Station.h"

        Default_Station::~Default_Station()
        {
                // does nothing
        }

        Default_Station::Default_Station(ostream& log_to,Nodetype* connection_node)
        {
                my_connection_node = connection_node;
                destination = log_to;
        }

        Default_Station::Default_Station(ostream& log_to)
        {
                my_connection_node = 0;
                destination = log_to;
        }


        Result_Code* Default_Station::Receive_Data(Sim_Packet* the_Data)
        {
                log("A default station got the data","Default_Station Message");
                return 0;
        }

        Result_Code* Default_Station::Send_Data(Sim_Packet* the_Data)
        {
                log("A default station was told to send data","Default_Station Message
");
                return gSimulation_Agent->Send_Event(
                        new System_Event(receive_data_event,the_Data),
                        my_connection_node);

        }

        Result_Code* Default_Station::Break_Connection(void)
        {
                log("A default station was told to break it's connection",
                        "Default_Station_Message");
                my_connection_node = 0;
                return 0;
        }

        Result_Code* Default_Station::Enable_Receive(void)
        {
                log("A default station was told to enable receive","Default_Station Me
ssage");
                return 0;
        }

        Result_Code* Default_Station::Disable_Receive(void)
        {
                log("A default station was told to disable receive","Default_Station M
essage");
                return 0;
        }

        Result_Code* Default_Station::New_Connection_Node(Nodetype* my_new_connection)
        {
                log("A default station was told to create a new connection","Default_S
tation Message");
                // in addition to worrying about reference counts, I think we should
                // worry also about creating a new connection between a station
                // and another node
                my_connection_node = my_new_connection;
                // but for now we aren't worrying about it.
                return 0;
        }
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Default_Station.h

        An off-the-shelf working Station subclass.

  To Do:
        I need to make sure that this deals with the reference counts of
        my_connection_node properly.
*/

#ifndef DEFAULT_STATION_H

#define DEFAULT_STATION_H

#include "Station.h"
#include "Logger.h"

class Default_Station : public Station,
                                        public Logger
{
  protected:
        ~Default_Station();

        Nodetype* my_connection_node;

  public:
        Default_Station(ostream& log_to,Nodetype* connection_node);

        Default_Station(ostream& log_to);


        virtual Result_Code* Receive_Data(Sim_Packet* the_Data);

        virtual Result_Code* Send_Data(Sim_Packet* the_Data);

        virtual Result_Code* Break_Connection(void);

        virtual Result_Code* Enable_Receive(void);

        virtual Result_Code* Disable_Receive(void);

        virtual Result_Code* New_Connection_Node(Nodetype* my_new_connection);

};

#endif
```

```
#include "Event_Queue.h"

        Event_Queue::Event_Queue()
        {

                // a do nothing constructor

        }

        bool Event_Queue:: not_empty()
        {
                return (!(pq.empty()));
        }

        void Event_Queue::push(Simulation_Event * sim_evnt_item)
        {
                pq.push(sim_evnt_item);

        }

        Simulation_Event * Event_Queue::pop()
        {
                Simulation_Event * t;
                if (pq.empty()) return 0; // maybe I should do a throw here?
                t = pq.top();
                pq.pop();
                return(t);
        }
```

```
#ifndef EVENT_QUEUE_H

#include <bool.h>
#include <function.h>
#include <deque.h>

#include <stack.h>
// supposedly stack.h is the one that has the priority_queue template in it

#include "Simulation_Event.h"

#define EVENT_QUEUE_H

// presently unimplemented
// an old version that I don't know if it works
//  can be found in old/

class Event_Queue {
  private:

  protected:
        priority_queue< deque<Simulation_Event*>, greater<Simulation_Event*> > pq;

  public:

        Event_Queue();
        bool not_empty();
        void push(Simulation_Event * sim_evnt_item);
        Simulation_Event * pop();
};

#endif
```

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

#include "Event_Receiver.h"

Event_Receiver::Event_Receiver()
{

}
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Event_Receiver.h
*/

#ifndef EVENT_RECEIVER_H

#define EVENT_RECEIVER_H

#include "config.h"
#include "Simulation_Event.h"
#include "Result_Code.h"

class Event_Receiver {

  private:

  protected:

  public:

        Event_Receiver(void);
        virtual Result_Code* Handle_Event(Simulation_Event* the_Event) = 0;

};

#endif
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Event_Sender.C
*/

#include "Event_Sender.h"

Result_Code* Event_Sender::Send_Event(
                Simulation_Event* the_Event,
                char* Receiver_Name
                ) // sends the event to the reciever indicated by the name
{
return the_Event_Receivers.lookup(vec(Receiver_Name))->Handle_Event(the_Event);
// eventually want Simulator_Agent to queue this
// but this should work for now as a quick hack
}

bool Event_Sender::Add_Event_Receiver(
                Event_Receiver& another_Event_Receiver,
                char* Receiver_Name
                ) // adds to the list of possible recievers "another_Event_Receiver"
                // returns 1 if the addition occured, 0 if the name already existed
{
return the_Event_Receivers.add(&another_Event_Receiver,vec(Receiver_Name));
// again, another quick hack
// (the pairing class hasn't been written yet, has it?)
// yet this should reduce our link errors
}

bool Event_Sender::Remove_Event_Receiver(char* Receiver_Name)
                // removes from the list of possible recievers "Receiver_Name"
                // returns 1 if the removal occured, 0 if no such receiver existed
{
return the_Event_Receivers.remove(vec(Receiver_Name));
// again, another quick hack...
}

Event_Sender::Event_Sender() // the constructor
{
// nothing for this constructor to do,
// as far as I can tell
}
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Event_Sender.h
*/

#ifndef EVENT_SENDER_H

#define EVENT_SENDER_H

#include <vector.h>
#include "config.h"
#include "Simulation_Event.h"
#include "Result_Code.h"
#include "Event_Receiver.h"
#include "pairings.h"
#include "RTSLUtils.h"

class Event_Sender {

  private:
        pairings<Event_Receiver*,vector<char> > the_Event_Receivers;
                // a list of all the Event Receivers this sender can send to,
                // along with their "names" whereby they are identified

  protected:
        virtual Result_Code* Send_Event(
                Simulation_Event* the_Event,
                char* Receiver_Name
                ); // sends the event to the reciever indicated by the name
        virtual bool Add_Event_Receiver(
                Event_Receiver& another_Event_Receiver,
                char* Receiver_Name
                ); // adds to the list of possible recievers "another_Event_Receiver"
                // returns 1 if the addition occured, 0 if the name already existed
        virtual bool Remove_Event_Receiver(char* Receiver_Name);
                // removes from the list of possible recievers "Receiver_Name"
                // returns 1 if the removal occured, 0 if no such receiver existed
  public:
        Event_Sender(); // the constructor
};

#endif
```

```
#include "Simulator_Agent.h"

Simulator_Agent* gSimulation_Agent;
```

```
#include "Simulator_Agent.h"

extern Simulator_Agent* gSimulation_Agent;
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Logger.C

  Note:
        This should be a parent class of any class that wants to
        access a log or log files.
*/

#include "Logger.h"

//      ostream& destination;

Logger::Logger(void) :destination(cerr)
{
        // this should be protected, and do nothing
}

Logger::Logger(Logger &the_Logger)      :destination(the_Logger.destination) // copy c
onstructor
{
}

Logger::Logger(ostream_withassign &log_to)      :destination(log_to)    // constructor
, takes where to log to as argument
{
}

bool Logger::log(char* something_to_log,char* category)
        // log logs the something_to_log as of type category to the log file.
        // For example, if the something_to_log was "execution table invalid"
        // and the category was "Error"
        // then the line "Error: execution table invalid"
        // would be appended to the log file
        /* a la the code
              destination << category << ": " << something_to_log << endl;
        */
        // returns 1 is succesfull,
        // 0 if either pointer argument is null,
        // or if the ostream was closed or had some other error.
{
        if (!(something_to_log || category)) return 0;
        destination << (category?category:"No category") << ": "
        << (something_to_log?something_to_log:"Nothing to log") << endl;
        return 1;
        // still need to put in code to check and see
        // if the logging actually went through well
}

bool Logger::log(Result_Code* aResultCode,char* category)
        // log logs the description string as of type category to the log file.
        // For example, if the description string was "execution table invalid"
        // and the category was "Error"
        // then the line "Error: execution table invalid"
        // would be appended to the log file
        /* a la the code
              destination << category << ": " << aResultCode.thy_string() << endl;
        */
        // returns 1 is succesfull,
        // 0 if either pointer argument is null,
        // or if the ostream was closed or had some other error
        // or if "thy_string" threw an error.
{
        if (!(aResultCode || category)) return 0;
        destination << (category?category:"No category") << ": "
        << (aResultCode?aResultCode->thy_string():"No Result Code to log") << endl;
        return 1;
        // still need to put in code to check and see
        // if the logging actually went through well
}

bool Logger::error(Result_Code* aResultCode)
        /* just like the code
                log(aResultCode,"Error");
        */
{
        return log(aResultCode,"Error");
}

bool Logger::warning(Result_Code* aResultCode)
        /* just like the code
                log(aResultCode,"Warning");
        */
{
        return log(aResultCode,"Warning");
}

bool Logger::die(Result_Code* aResultCode)
        /* just like the code
                log(aResultCode,"Fatal");
                exit(aResultCode.thy_id_number());
        */
        // designed to emulate the effect of perl's "die" command
{
                bool result;
                result = log(aResultCode,"Fatal");
                exit(aResultCode->thy_id_number());
                return result;
                // need to change prototype - this should be declared void, right?
}

bool Logger::error(char* some_error)
        /* just like the code
                log(some_error,"Error");
        */
{
        return log(some_error,"Error");
}

bool Logger::warning(char* some_warning)
        /* just like the code
                log(some_warning,"Warning");
        */
{
        return log(some_warning,"Warning");
}

bool Logger::die(char* something_fatal)
        /* just like the code
                log(something_fatal,"Fatal");
                exit(1);
        */
        // designed to emulate the effect of perl's "die" command
{
                bool result;
                result = log(something_fatal,"Fatal");
                exit(1);
                return result;
                // need to change prototype - this should be declared void, right?
}
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Logger.h

  Note:
        This should be a parent class of any class that wants to
        access a log or log files.
*/

#ifndef LOGGER_H

#define LOGGER_H

#include "config.h"
#include <iostream.h>
#include "Result_Code.h"

class Logger {

  protected:
        ostream_withassign& destination;


  public:
        Logger();
        Logger(Logger &the_Logger);     // copy constructor
        Logger(ostream_withassign &log_to);     // constructor, takes where to log to
as argument

        bool log(char* something_to_log,char* category);
                // log logs the something_to_log as of type category to the log file.
                // For example, if the something_to_log was "execution table invalid"
                // and the category was "Error"
                // then the line "Error: execution table invalid"
                // would be appended to the log file
                /* a la the code
                        destination << category << ": " << something_to_log << endl;
                */
                // returns 1 is succesfull,
                // 0 if either pointer argument is null,
                // or if the ostream was closed or had some other error.

        bool log(Result_Code* aResultCode,char* category);
                // log logs the description string as of type category to the log file

                // For example, if the description string was "execution table invalid

                // and the category was "Error"
                // then the line "Error: execution table invalid"
                // would be appended to the log file
                /* a la the code
                        destination << category << ": " << aResultCode.thy_string() <<
 endl;
                */
                // returns 1 is succesfull,
                // 0 if either pointer argument is null,
                // or if the ostream was closed or had some other error
                // or if "thy_string" threw an error.

        bool error(Result_Code* aResultCode);
                /* just like the code
                        log(aResultCode,"Error");
                */

        bool warning(Result_Code* aResultCode);
                /* just like the code
                        log(aResultCode,"Warning");
                */

        bool die(Result_Code* aResultCode);
                /* just like the code
                        log(aResultCode,"Fatal");
                        exit(aResultCode.thy_id_number());
                */
                // designed to emulate the effect of perl's "die" command

        bool error(char* some_error);
                /* just like the code
                        log(some_error,"Error");
                */

        bool warning(char* some_warning);
                /* just like the code
                        log(some_error,"Warning");
                */

        bool die(char* something_fatal);
                /* just like the code
                        log(some_error,"Fatal");
                        exit(1);
                */
};

#endif
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Medium.h

*/

#ifndef MEDIUM_H

#define MEDIUM_H

#include "config.h"
#include "Event_Sender.h"
#include "Event_Receiver.h"

class Medium :  public Event_Receiver,
                                        public Event_Sender
{
  protected:
        ~Medium();

  public:
        Medium();

        virtual Result_Code* Handle_Event(Simulation_Event* the_Event);

};

#endif
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Network.h

*/

#ifndef NETWORK_H

#define NETWORK_H

#include "config.h"
#include "Event_Sender.h"
#include "Event_Receiver.h"

class Network : public Event_Receiver,
                                        public Event_Sender
{
  protected:
        ~Network();

  public:
        Network();

        virtual Result_Code* Handle_Event(Simulation_Event* the_Event);

};

#endif
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Nodetype.h

*/

#ifndef NODETYPE_H

#define NODETYPE_H

#include "config.h"
#include "Event_Sender.h"
#include "Event_Receiver.h"
#include "Result_Code.h"
#include "Control_Region.h"
#include "Sim_Packet.h"

class Nodetype :        public Event_Receiver,
                                public Event_Sender
{
  protected:
        ~Nodetype()
        {
        };

  public:
        Nodetype()
        {
        }
        ;

        virtual Result_Code* Handle_Event(Simulation_Event* the_Event) = 0;

        virtual Result_Code * Connect_Control_Region(Control_Region*) = 0;
    virtual Result_Code * Disconnect_Control_Region(Control_Region*) = 0;
    virtual Result_Code * Observe_Begin_Transmit(Sim_Packet*) = 0;
    virtual Result_Code * Observe_End_Transmit(Sim_Packet*) = 0;
    virtual Result_Code * Observe_Abort_Trasmit(Sim_Packet*) = 0;

};

#endif
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Protocol.h

*/

#ifndef PROTOCOL_H

#define PROTOCOL_H

#include "config.h"
#include "Ref_Cnt_Obj.h"
#include "Event_Sender.h"
#include "Event_Receiver.h"

class Protocol :        public Event_Receiver,
                                public Event_Sender,
                                public Ref_Cnt_Obj
{
  protected:
        ~Protocol();

  public:
        Protocol();

        virtual Result_Code* Handle_Event(Simulation_Event* the_Event);

};

#endif
```

```
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <assert.h>
#include <stdio.h>
#include "RTSLUtils.h"
#include "System_Event.h"

int die(char* death_string)
{
        cerr << death_string;
        exit(1);
        return 0;
}

vector<char> vec(char * s)
        // Return vector<char> containing the characters of s
        // (not including the terminating null).
{
        vector<char> x;
        while (*s != '\0') {
                x.push_back(*s++);

        };
        return x;
}
```

```
#include <vector.h>
#include "Event_Queue.h"

int die(char*);

vector<char> vec(char * s);
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Ref_Cnt_Obj.C

  Notes:

        Reference Count Objects can only be created via pointers;
        The code used to deal with the Reference Count Object must manually
                1) keep track of when to increment/decrement the count
                2) be very careful that if the controlling/using object is duplicated
                        that the resulting counts are kept accurate.
*/

#include "Ref_Cnt_Obj.h"

Ref_Cnt_Obj::Ref_Cnt_Obj()
{
        my_count = 0;
// I'll figure out what (if anything) this is supposed to do later.
}

Ref_Cnt_Obj::~Ref_Cnt_Obj()
{
// presently a do nothing destructor -
// we should, however, implement error checking
// to make sure that we're not destroying this
// when something is actually referencing it
}

Result_Code * Ref_Cnt_Obj::Destroy(Ref_Cnt_Obj* to_die)
{
        delete to_die;
        return 0;
}

Result_Code * Ref_Cnt_Obj::Decrement_Count(Ref_Cnt_Obj* to_decrement)
{
        (to_decrement->my_count)--;
        if (to_decrement->my_count == 0) delete to_decrement;
        return 0;
}

Result_Code * Ref_Cnt_Obj::Increment_Count(Ref_Cnt_Obj* to_increment)
{
        (to_increment->my_count)++;
        return 0;
}
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Ref_Cnt_Obj.h

  Notes:

        Reference Count Objects can only be created via pointers;
        The code used to deal with the Reference Count Object must manually
                1) keep track of when to increment/decrement the count
                2) be very careful that if the controlling/using object is duplicated
                        that the resulting counts are kept accurate.
*/

#ifndef REF_CNT_OBJ_H

#define REF_CNT_OBJ_H

#include "config.h"
#include "Result_Code.h"

class Ref_Cnt_Obj {

  private:

  protected:
        unsigned long my_count;
        ~Ref_Cnt_Obj();

  public:
        Ref_Cnt_Obj();
        static Result_Code * Destroy(Ref_Cnt_Obj* to_die);
        static Result_Code * Decrement_Count(Ref_Cnt_Obj* to_decrement);
        static Result_Code * Increment_Count(Ref_Cnt_Obj* to_increment);

};

#endif
```

```
#include <string.h>
#include <bool.h>
#include "Result_Code.h"

// the static data members

pairings<char*,unsigned long> Result_Code::Result_Codes;

                // the Result_Code class has a single
                // associative array of id numbers and
                // description strings,
                // for referencing a description of the
                // instance's value's meaning or
                // for automatic creation of id numbers
                // based on description strings that
                // one wants to _dynamically_ enter into
                // the associative array.


unsigned long Result_Code::curr_id_max = 0;

                // at any given point during execution
                // is an upper bound for the
                // id numbers in the associative array.

// the non-static member functions

Result_Code::Result_Code()
{
}

Result_Code::Result_Code(char * arg) // must be a string constant that is arg
{
        bool done = false;
        for (int i = 0; i < curr_id_max; i++) {
                if (0 == strcmp(Result_Codes.lookup(i),arg)) {
                        my_id_number = i;
                        done = true;
                        break;
                };
        };
        if (!done) {
                Result_Codes.add(arg,curr_id_max);
                my_id_number = curr_id_max;
                curr_id_max++;
        };
        // there's got to be a better way to implement this
        // but this is the best that I can think of for now
}

char* Result_Code::thy_string(void)
{
        return Result_Codes.lookup(my_id_number);
}

unsigned long Result_Code::thy_id_number(void)
{
        return my_id_number;
}
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Result_Code.h

  Notes:

        a 0 for a Result_Code * should be interpretted as ok
        (or as execution should never have reached that point).
        "ok" will be the standard "ok" message string.

*/

#ifndef RESULT_CODE_H

#define RESULT_CODE_H

#include "config.h"

#include "pairings.h"

#include <bool.h>

class Result_Code {

  private:

  protected:

        unsigned long my_id_number;

                // each Result_Code instance has an id number
                // that represents it's value.


        static pairings<char*,unsigned long> Result_Codes;

                // the Result_Code class has a single
                // associative array of id numbers and
                // description strings,
                // for referencing a description of the
                // instance's value's meaning or
                // for automatic creation of id numbers
                // based on description strings that
                // one wants to _dynamically_ enter into
                // the associative array.


        static unsigned long curr_id_max ;

                // at any given point during execution
                // is an upper bound for the
                // id numbers in the associative array.


        bool query_id_number_exists(unsigned long);

                // determines whether or not the given
                // id number exists in the associative array
                // of id numbers and description strings.

        bool query_desc_string_exists(char*);

                // determines whether or not the given
                // description string exists in the associative
                // array of id numbers and description strings.


        unsigned long id_number_of_desc_string(char*);

                // looks up in the associative array of
                // id numbers and description strings the
                // id number corresponding to the description
                // string that is it's argument.
                // Throws an error if the given string is not found.


        unsigned long unique_id_number(void);

                // returns an id number that presently is not
                // in the associative array.


        char* desc_string_of_id_number(unsigned long);

                // looks up in the associative array of
                // id numbers and description strings the
                // description string corresponding to
                // the id number that is it's argument.
                // Allocates new memory for the string,
                // copies it over, and returns a pointer to
                // the copy, to prevent unauthorized access to
                // the description strings.
                // Throws an error if the id number is not found.


        bool add_pairing(unsigned long, char*);

                // adds the given pairing represented by the arguments
                // to the associative array of description strings
                // and id numbers.
                // May or may not perform checks to maintain internal
                // consistancy.
                // If it does, throws errors when serious problems arise.
                // If the operation was successful, a non-zero value is returned.
                // If the operation was unsuccessful, but no serious errors
                // occurred, then a zero value is returned.


public:
        Result_Code(char* description_string);

                // normal constructor for the Result_Code class
                /*      automatic creation and management of id numbers,
                        checks argument to see if it exists,
                        if not adds it with new id number,
                        otherwise just sets id numebr correctly.
                 */


        Result_Code(Result_Code& the_Result_Code);

                // copy constructor for Result_Code class

        Result_Code(void);
        // default constructor


Result_Code(    unsigned long desired_id_number,
                        char* description_string,
                        bool& completion);

        /*      constructor that tries to add the given
                id number and description string pairing
                to the associative array.
                The reference to the boolean value completion
                is where it will store whether or not
                it succeded in adding that description string and
                id number to the associative array.
                At present, it's behavior if it can't
                because the id_number is already taken is
                to not change the associative array at all
                and completion will be zero.
                However, if the id_number is not already taken
                but the description string is,
                the behavior of this constructor is undefined.
                --read as, there may be associative array inconsistancies
                        or a call to assert may be made to terminate execution
                        or completion may be zero,
                        or an error may be thrown
        */


const Result_Code& operator=(Result_Code& another_Result_Code);

        // assigns the other instance's value to the
        // current instance's value.
        // May or may not check to see that the value actually exists
        // in the associative array.


const bool operator==(Result_Code& another_Result_Code);

        // Compares two instances to see if their values
        // are the same.
        // May or may not check to see if the value actually exists
        // in the associative array.


char* thy_string(void);

        // returns the description string corresponding to
        // the id number of the current instance.
        // If the instance is invalid and the id number
        // doesn't actually exist in the associative array,
        // then an error is thrown.

unsigned long thy_id_number(void);

        // returns the id number of the current instance.
};

#endif
```

```
#include "Data_Obj.h"
#include "Ref_Cnt_Obj.h"

#ifndef SIM_PACKET_H
#define SIM_PACKET_H

class Sim_Packet : public Data_Obj, public Ref_Cnt_Obj
{
  public:

        Sim_Packet()
        // do nothing constructor
        {
        };
};

#endif
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Simulation_Event.C
*/

#include "Simulation_Event.h"

Simulation_Event::Simulation_Event(void)
{
        my_count = 1; // as this is a Ref_Cnt_Obj, should initilize my_count properly
}
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Simulation_Event.h
*/

#ifndef SIMULATION_EVENT_H

#define SIMULATION_EVENT_H

#include "config.h"
#include "Ref_Cnt_Obj.h"
#include "TimeStamp_Obj.h"

class System_Event;
class Placeholder_Event;

class Simulation_Event : public Ref_Cnt_Obj, public TimeStamp_Obj {

  private:

  protected:

  public:
        Simulation_Event(); // the constructor
        virtual System_Event* as_System_Event() = 0;
                        // more elegant than downcasting
                        // returns a null pointer if it is not
                        // a system event
        virtual Placeholder_Event* as_Placeholder_Event() = 0;
                        // more elegant than downcasting
                        // returns a null pointer if it is not a placeholder event
};

static inline bool operator<(Simulation_Event* a,Simulation_Event* b)
{
        return (*a < *b);
}

static inline bool operator>(Simulation_Event* a,Simulation_Event* b)
{
        return (*a > *b);
}

#endif
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Simulator_Agent.h

*/

#include <fstream.h>
#include "Simulator_Agent.h"
#include "Event_Queue.h"
#include "RTSLUtils.h"
#include "System_Event.h"
#include "Sim_Packet.h"

Simulator_Agent::~Simulator_Agent()
{
        // destroys the event queue
        delete my_event_queue;
}

Simulator_Agent::Simulator_Agent(char* the_file)
{
        my_event_queue = new Event_Queue;

        ReadEDF(the_file); //Read_EDF reads the file
        // and places everything in the file in the event queue of
        // Simulator_Agent
}

void Simulator_Agent::ReadEDF(char* filename)
{
        // this still needs some work, but at least it's no longer a dummy function
        unsigned long how_many_events,i;
        long temp_event_code;
        ifstream * the_file;
        assert(the_file = new ifstream(filename));
        (*the_file) >> how_many_events;
        for(i = 0; i < how_many_events; i++) {
                (*the_file) >> temp_event_code;
// teh following if statement is a hack for now
// this whole thing I intend to replace my a more
// elegant EDF format and reader
                if ((temp_event_code != send_data_event)&&
                        (temp_event_code != receive_data_event)) {
                        my_event_queue->push(new System_Event(temp_event_code));
                } else {
                        my_event_queue->push(new System_Event(temp_event_code,
                                new Sim_Packet));
                };
        };
}

Simulator_Agent::Do_Simulation()
{
        bool ok = true;
        while (my_event_queue->not_empty() && ok) {
                ok = Process_Event(my_event_queue->pop());
        }
        return ok;
}

bool Simulator_Agent::Process_Event(Simulation_Event* the_Event)
{
        System_Event * sys_evnt_ptr;

        if (the_Event == 0) {
                return false;
        } else {
                if (sys_evnt_ptr = the_Event->as_System_Event()) {
                        return Process_Event(sys_evnt_ptr);
                } else {
                        die("Hey, this shouldn't be happening!\n");
                        return false;
                };
        };
}

Result_Code * Simulator_Agent::Send_Event(Simulation_Event * event,Event_Receiver * de
stination)
{
        return destination->Handle_Event(event);
}
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Simulator_Agent.h

*/

#ifndef SIMULATOR_AGENT_H

#define SIMULATOR_AGENT_H

#include "config.h"
#include "Event_Sender.h"
#include "Event_Queue.h"
#include "Nodetype.h"

class Simulator_Agent : public Event_Sender
{
  protected:
        ~Simulator_Agent();
        Event_Queue * my_event_queue;

  public:
        Simulator_Agent(char*);
        int Do_Simulation();
        bool Process_Event(Simulation_Event*);
        virtual bool Process_Event(System_Event*) =0;
        virtual Result_Code * Send_Event(Simulation_Event *,Event_Receiver *);
        virtual Nodetype* Name2Node(char*) =0;
        void ReadEDF(char*);
};

#endif
```

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <string.h>

#include "Station.h"
#include "Globals.h"

#include "Simulation_Event.h"
#include "System_Event.h"
#include "Sim_Packet.h"

static char * const_test_event = "Test Event";

Result_Code* Station::Handle_Event(Simulation_Event* the_Event)
{
        System_Event * sys_evnt_ptr;
        pair<int,char**> * its_args;

        sys_evnt_ptr = 0;
        if (the_Event && (sys_evnt_ptr = the_Event->as_System_Event())) {
                switch (sys_evnt_ptr->thy_Event_Code()) {
                  case send_self_test_event:
                        return  gSimulation_Agent->Send_Event(new System_Event(a_test_
event),this);
                        break;
                  case a_test_event:
                        Test_Event_Received();
                        return 0;
                        break;
                  case receive_data_event:
                        return Receive_Data(sys_evnt_ptr->get_Packet());
                  case send_data_event:
                        return Send_Data(sys_evnt_ptr->get_Packet());
                  case break_connection_event:
                        return Break_Connection();
                  case enable_receive_event:
                        return Enable_Receive();
                  case disable_receive_event:
                        return Disable_Receive();
                  case new_connection_node_event:
                        return New_Connection_Node(sys_evnt_ptr->get_Node());
                  case arg_based_event:
                        its_args = &(sys_evnt_ptr->thy_Args());
                        switch (its_args->first) {
                          case 1:
                                if (0 == strcmp(its_args->second[0],const_test_event))
 {
                                        Test_Event_Received();
                                        return 0;
                                } else if (0 == strcmp(its_args->second[0],"Send Self
Test Event")) {
                                        return
                                                gSimulation_Agent->Send_Event(
                                                        new System_Event(
                                                                make_pair(1,&const_tes
t_event)
                                                        ),
                                                        this
                                                );
                                } else if (0 == strcmp(its_args->second[0],"Send Self
Test Packet")) {
                                        return
                                                gSimulation_Agent->Send_Event(
                                                        new System_Event(receive_data_
event,new Sim_Packet),this
                                                        );
                                } else if (0 == strcmp(its_args->second[0],"Break Conn
ection")) {
                                        return Break_Connection();
                                } else if (0 == strcmp(its_args->second[0],"Enable Rec
eive")) {
                                        return Enable_Receive();
                                } else if (0 == strcmp(its_args->second[0],"Disable Re
ceive")) {
                                        return Disable_Receive();
                                } else die("Apparently Invalid Argument Based Event Ha
ndling.\n"
                                                "argc = 1\n");
                          case 2:
                                if (0 == strcmp(its_args->second[0],"New Connection No
de")) {
                                        return
                                                New_Connection_Node(
                                                        gSimulation_Agent->Name2Node(i
ts_args->second[1])
                                                );
                                } else die("Apparently Invalid Argument Based Event Ha
ndling.\n"
                                                "argc = 2\n");
                          default:
                                die("Apparently Invalid Argument Based Event Handling.
\n"
                                        "argc != 1, argc != 2\n");
                        }
                        assert(0);
                        return(0);
                        break;

                default:
                        assert(0);
                        return 0;
                };
        };
}

Station::Station(void)
{
// do nothing, as far as I can tell
}

Station::Station(Nodetype* my_connection_node)
{
        New_Connection_Node(my_connection_node);
}

Station::~Station(void)
{
        // do nothing, as far as I can tell
}

void Station::Test_Event_Received(void) {
        cout << "Test Event Received" << endl;
        // this was just a hack for the demo
}
```

```
/*
	Run-Time Sumulation Library version 0.01pa
	(c) 1996 Andrew Chen

	No guarrantees.

	Station.h

	This is an abstract class.
	For an off-the-shelf working Station subclass, use Default_Station.
*/

#ifndef STATION_H

#define STATION_H

#include "config.h"
#include "Event_Sender.h"
#include "Event_Receiver.h"
#include "Sim_Packet.h"
#include "Nodetype.h"

class Station : public Event_Receiver,
					public Event_Sender
{
  protected:
	~Station();


  public:
	Station(Nodetype* my_connection_node);
	Station();

	virtual Result_Code* Handle_Event(Simulation_Event* the_Event);
	virtual Result_Code* Receive_Data(Sim_Packet* the_Data) =0;
	virtual Result_Code* Send_Data(Sim_Packet* the_Data) =0;
	virtual Result_Code* Break_Connection(void) =0;
	virtual Result_Code* Enable_Receive(void) =0;
	virtual Result_Code* Disable_Receive(void) =0;
	virtual Result_Code* New_Connection_Node(Nodetype* my_new_connection) =0;
	void Test_Event_Received(void);

};

#endif
```

```
#include "Station2Station_Connection.h"

#include "System_Event.h"
#include "Globals.h"

#include "Default_Station.h"
// that's just a hack

Station2Station_Connection::~Station2Station_Connection()
{
}


Station2Station_Connection::Station2Station_Connection():other_station((Station*)0)
{
}

Result_Code * Station2Station_Connection::Attach_to(Station * some_station)
{
        if (my_station) {
                assert(!other_station);
                other_station = some_station;
        } else {
                my_station = some_station;
        };
        return (new Result_Code("ok"));
}

Result_Code * Station2Station_Connection::Handle_Event(Simulation_Event* the_Event)
{
        // presently this function is a hack for the demo
        // as are all of the following

        System_Event * the_sys_evnt;

        assert(the_sys_evnt = the_Event->as_System_Event());
        assert(
                (the_sys_evnt->thy_Event_Code() == send_data_event) ||
                (the_sys_evnt->thy_Event_Code() == receive_data_event)
                );
        assert(my_station);
        if (!other_station) {
                other_station = new Default_Station(cerr,this);
                assert(other_station);
        };
        return gSimulation_Agent->Send_Event(
                new System_Event(
                        the_sys_evnt->thy_Event_Code(),
                        the_sys_evnt->get_Packet()
                ),
                other_station
        );
        // so as I understand it, in the demo the send data event gets sent to the fir
st station,
        // which creates a packet and sends it to the Station2Station_Connection node,
        // which creates another station and sends it there,
        // which then sends a receive data event (w/ packet) to the s2s_c_n which then
 sends it back.
}

Result_Code * Station2Station_Connection::Connect_Control_Region(Control_Region*)
{
        die("Station2Station_Connection::Connect_Control_Region(Control_Region*)\n"
                "unimplemented.\n");
        return 0;
}

Result_Code * Station2Station_Connection::Disconnect_Control_Region(Control_Region*)
{
        die("Station2Station_Connection::Disconnect_Control_Region(Control_Region*)\n"
                "unimplemented.\n");
        return 0;
}

Result_Code * Station2Station_Connection::Observe_Begin_Transmit(Sim_Packet*)
{
        die("Station2Station_Connection::Observe_Begin_Transmit(Sim_Packet*)\n"
                "unimplemented.\n");
        return 0;
}

Result_Code * Station2Station_Connection::Observe_End_Transmit(Sim_Packet*)
{
        die("Station2Station_Connection::Observe_End_Transmit(Sim_Packet*)\n"
                "unimplemented.\n");
        return 0;
}

Result_Code * Station2Station_Connection::Observe_Abort_Trasmit(Sim_Packet*)
{
        die("Station2Station_Connection::Observe_Abort_Trasmit(Sim_Packet*)\n"
                "unimplemented.\n");
        return 0;
}
```

```
// not presently debugged nor tested
#ifndef STATION2STATION_CONNECTION_H

#define STATION2STATION_CONNECTION_H
#include "Station_Connection.h"

class Station2Station_Connection : public Station_Connection
{
  protected:
        ~Station2Station_Connection();
        Station* other_station;

  public:
        Station2Station_Connection();

        virtual Result_Code * Handle_Event(Simulation_Event* the_Event);
        virtual Result_Code * Attach_to(Station* some_station);
        virtual Result_Code * Connect_Control_Region(Control_Region*);
    virtual Result_Code * Disconnect_Control_Region(Control_Region*);
    virtual Result_Code * Observe_Begin_Transmit(Sim_Packet*);
    virtual Result_Code * Observe_End_Transmit(Sim_Packet*);
    virtual Result_Code * Observe_Abort_Trasmit(Sim_Packet*);

};

#endif
```

```
#include "Station_Connection.h"

Station_Connection::~Station_Connection()
{
}

Station_Connection::Station_Connection():my_station((Station *)0)
{
}

Result_Code * Station_Connection::Handle_Event(Simulation_Event* the_Event)
{
        // this function is incomplete and unimplemented
        // presently, as a hack, it is exprected to be over-ridden
        // ideally in the complete implementation this should not be necessary
        die("Station_Connection::Handle_Event(Simulation_Event* the_Event)\n"
                "not implemented yet.\n");
        return 0;
}

Result_Code * Station_Connection::Attach_to(Station* some_station)
{
        assert(!my_station);
        my_station = some_station;
        return new Result_Code("ok");
}
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Station_Connection.h

  Note:

        A station connection node can only be connected to one station at a time.
*/

#ifndef STATION_CONNECTION_H

#define STATION_CONNECTION_H

#include "config.h"
#include "Event_Sender.h"
#include "Event_Receiver.h"
#include "Nodetype.h"
#include "Station.h"

class Station_Connection :      public Nodetype
{
  protected:
        ~Station_Connection();
        Station * my_station;

  public:
        Station_Connection();

        virtual Result_Code * Handle_Event(Simulation_Event* the_Event);
        virtual Result_Code * Attach_to(Station* some_station);
        virtual Result_Code * Connect_Control_Region(Control_Region*) = 0;
    virtual Result_Code * Disconnect_Control_Region(Control_Region*) = 0;
    virtual Result_Code * Observe_Begin_Transmit(Sim_Packet*) = 0;
    virtual Result_Code * Observe_End_Transmit(Sim_Packet*) = 0;
    virtual Result_Code * Observe_Abort_Trasmit(Sim_Packet*) = 0;

};

#endif
```

```
#include "Station_Connection_Echo.h"

#include "System_Event.h"
#include "Globals.h"

Station_Connection_Echo::~Station_Connection_Echo()
{
}


Station_Connection_Echo::Station_Connection_Echo()
{
}


Result_Code * Station_Connection_Echo::Handle_Event(Simulation_Event* the_Event)
{
        // presently this function is a hack for the demo
        // as are all of the following

        System_Event * the_sys_evnt;

        assert(the_sys_evnt = the_Event->as_System_Event());
        assert(
                (the_sys_evnt->thy_Event_Code() == send_data_event) ||
                (the_sys_evnt->thy_Event_Code() == receive_data_event)
                );
        assert(my_station);
        return gSimulation_Agent->Send_Event(
                new System_Event(
                        the_sys_evnt->thy_Event_Code(),
                        the_sys_evnt->get_Packet()
                ),
                my_station
        );
}

Result_Code * Station_Connection_Echo::Connect_Control_Region(Control_Region*)
{
        die("Station_Connection_Echo::Connect_Control_Region(Control_Region*)\n"
                "unimplemented.\n");
        return 0;
}

Result_Code * Station_Connection_Echo::Disconnect_Control_Region(Control_Region*)
{
        die("Station_Connection_Echo::Disconnect_Control_Region(Control_Region*)\n"
                "unimplemented.\n");
        return 0;
}

Result_Code * Station_Connection_Echo::Observe_Begin_Transmit(Sim_Packet*)
{
        die("Station_Connection_Echo::Observe_Begin_Transmit(Sim_Packet*)\n"
                "unimplemented.\n");
        return 0;
}

Result_Code * Station_Connection_Echo::Observe_End_Transmit(Sim_Packet*)
{
        die("Station_Connection_Echo::Observe_End_Transmit(Sim_Packet*)\n"
                "unimplemented.\n");
        return 0;
}

Result_Code * Station_Connection_Echo::Observe_Abort_Trasmit(Sim_Packet*)
{
        die("Station_Connection_Echo::Observe_Abort_Trasmit(Sim_Packet*)\n"
                "unimplemented.\n");
        return 0;
}
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Station_Connection_Echo.h

  Note:

        A station connection node can only be connected to one station at a time.
*/

#ifndef STATION_CONNECTION_ECHO_H

#define STATION_CONNECTION_ECHO_H

#include "config.h"
#include "Event_Sender.h"
#include "Event_Receiver.h"
#include "Nodetype.h"
#include "Station.h"
#include "Station_Connection.h"

class Station_Connection_Echo : public Station_Connection
{
  protected:
        ~Station_Connection_Echo();

  public:
        Station_Connection_Echo();

        virtual Result_Code * Handle_Event(Simulation_Event* the_Event);
        virtual Result_Code * Connect_Control_Region(Control_Region*);
    virtual Result_Code * Disconnect_Control_Region(Control_Region*);
    virtual Result_Code * Observe_Begin_Transmit(Sim_Packet*);
    virtual Result_Code * Observe_End_Transmit(Sim_Packet*);
    virtual Result_Code * Observe_Abort_Trasmit(Sim_Packet*);

};

#endif
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        System_Event.h

*/

#ifndef SYSTEM_EVENT_H

#define SYSTEM_EVENT_H

#include <bool.h>
#include <vector.h>
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include "config.h"
#include "Simulation_Event.h"
#include "RTSLUtils.h"
#include "Sim_Packet.h"
#include "Nodetype.h"

// here are the defines for various event codes
// (this is presently just a hack for the demo prototype
// a more elegant system needs to be devised...)

#define send_self_test_event    1001
#define a_test_event                    1002
#define new_station                             1003
#define send_most_recently_created_station_test_event   1004
#define receive_data_event              1005
#define send_data_event                 1006
#define break_connection_event  1007
#define enable_receive_event    1008
#define disable_receive_event   1009
#define new_connection_node_event       1010
#define new_echo_node                   1011
#define new_s2s_conn_node               1012

#define arg_based_event                 1000
//hopefully arg_based_event is the more elegant system which we needed

class System_Event :    public Simulation_Event
{
  protected:
        long the_Event_Code;
        bool has_args;
        pair<int,char**>& args;
        Sim_Packet * the_Packet;
        Nodetype * the_Node;

        virtual ~System_Event()
        {
                if (the_Packet) Sim_Packet::Decrement_Count(the_Packet);
        };

  public:
        System_Event(long Event_Code) :
                the_Packet(0),
                the_Node(0),
                args(make_pair(0,(char**)0)) // arg is currently just a hack
        {// this whole function here is just a hack for the demo prototype
                the_Event_Code = Event_Code;
                has_args = false;
                assert(
                        (Event_Code != send_data_event)&&
                        (Event_Code != receive_data_event));
        };

        System_Event(long Event_Code,Sim_Packet * Packet) :
                the_Packet(Packet),
                the_Node(0),
                args(make_pair(0,(char**)0)) // arg is currently just a hack
        {// this whole function here is just a hack for the demo prototype
                the_Event_Code = Event_Code;
                has_args = false;
                assert(
                        (Event_Code == send_data_event)||
                        (Event_Code == receive_data_event));
                if (the_Packet) Sim_Packet::Increment_Count(the_Packet);
        };

        System_Event(long Event_Code,Nodetype * Node) :
                the_Packet(0),
                the_Node(Node),
                args(make_pair(0,(char**)0)) // arg is currently just a hack
        {// this whole function here is just a hack for the demo prototype
                the_Event_Code = Event_Code;
                has_args = false;
                assert(Event_Code == new_connection_node_event);
        };

        System_Event(pair<int,char**> &argargs):args(argargs)
        {
                has_args = true;
                args = argargs;
                // do I need that above line?
                the_Event_Code = arg_based_event;

        };

        Sim_Packet * get_Packet(void)
        {
                assert(the_Packet);
                return the_Packet;
        };

        Nodetype * get_Node(void)
        {
                assert(the_Node);
                return the_Node;
        };

        pair<int,char**>& thy_Args()
        {
                assert(has_args);
                return args;
        };

        long thy_Event_Code() {
                return the_Event_Code;
        }
        ;
        System_Event* as_System_Event()
        {
                return this;
        }
        ;
        Placeholder_Event* as_Placeholder_Event()
        {
                return 0;
        }
        ;
        void Destroy()
        {
                delete this;
        }
        ;
};

#endif
```

```
#include "TimeStamp_Obj.h"

TimeStamp_Obj::TimeStamp_Obj()
{
// do nothing constructor
}

bool const TimeStamp_Obj::operator<(const TimeStamp_Obj& other_time_stamp)
{
        return (my_time < other_time_stamp.my_time);
}

bool const TimeStamp_Obj::operator>(const TimeStamp_Obj& other_time_stamp)
{
        return (my_time > other_time_stamp.my_time);
}
```

```
// timestamp class - plug-in time-has-a-ordering

#ifndef TIMESTAMP_OBJ_H

#define TIMESTAMP_OBJ_H
#include "Time_Class.h"
#include <bool.h>

class TimeStamp_Obj {
  private:

  protected:
#ifdef SUPPORTS_MUTABLE
        mutable
#endif
        Time_Class my_time;

  public:

        TimeStamp_Obj();

        void operator+=(Time_Class& time_adj)
        // to augment it's current time-value
        {
                my_time += time_adj;
        };

        bool const operator<(const TimeStamp_Obj& other_time_stamp);
        bool const operator>(const TimeStamp_Obj& other_time_stamp);
        friend
#ifdef STATIC_AND_INLINE
static
#endif
        inline bool operator<(const TimeStamp_Obj& a,const TimeStamp_Obj& b);
        friend
#ifdef STATIC_AND_INLINE
static
#endif
        inline bool operator>(const TimeStamp_Obj& a,const TimeStamp_Obj& b);


};

#ifdef STATIC_AND_INLINE
static
#endif
inline bool operator<(const TimeStamp_Obj& a,const TimeStamp_Obj& b)
{
        return (a.my_time < b.my_time);
}

#ifdef STATIC_AND_INLINE
static
#endif
inline bool operator>(const TimeStamp_Obj& a,const TimeStamp_Obj& b)
{
        return (a.my_time > b.my_time);
}


#endif
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Time.C

  Note:

        The Time class can either be the time since a certain time,
        or a particular amount of elapsed time.
        The exact usage may vary throughout the code,
        as a consequence, this file may be incomplete.

*/

#include "Time_Class.h"
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

Time_Class::Time_Class()
{
        the_time = time(NULL);
}

void Time_Class::operator+=(Time_Class& time_adj)
        // to augment it's current time-value
        // - needs be modified if more than seconds are being kept track of
        {
                the_time += time_adj.the_time;
        };

bool Time_Class::operator==(Time_Class* a_time)
{
        if (a_time) {
                if (the_time == a_time->the_time) {
                        return true;
                } else {
                        return false;
                };
        } else {
                return false;
                // maybe we want to throw an error here ?
        };
}

bool Time_Class::operator<(Time_Class& a_time)
{
        return (the_time < a_time.the_time);
}

bool Time_Class::operator>(Time_Class& a_time)
{
        return (the_time > a_time.the_time);
}

void Time_Class::operator=(Time_Class* the_other_time)
{
// later on we want more robust error handling with throw statements and stuff
// but for now this should do.
assert(the_other_time);
//      if (the_other_time) {
                the_time = the_other_time->the_time;
//      } else {
}

void Time_Class::out(ostream& the_out)
{
        the_out << ctime(&the_time);
}
```

```
/*
        Run-Time Sumulation Library version 0.01pa
        (c) 1996 Andrew Chen

        No guarrantees.

        Time.h

  Note:

        The Time_Class class can either be the time since a certain time,
        or a particular amount of elapsed time.
        The exact usage may vary throughout the code,
        as a consequence, the header here may be incomplete.

*/

#ifndef TIME_H

#define TIME_H

#include "config.h"
#include <bool.h>
#include <time.h>
#include <iostream.h>

class Time_Class {

  private:

  protected:
        time_t  the_time;

  public:
        Time_Class();
        virtual bool operator==(Time_Class* a_time);
        virtual bool operator<(Time_Class& a_time);
        virtual bool operator>(Time_Class& a_time);
        virtual void operator+=(Time_Class& time_adj);

        virtual void operator=(
                Time_Class* the_other_time
                );
        virtual void out(ostream& the_out);

        friend
#ifdef STATIC_AND_INLINE
static
#endif
        inline bool operator<(const Time_Class& a,const Time_Class& b);
        friend
#ifdef STATIC_AND_INLINE
static
#endif
        inline bool operator>(const Time_Class& a,const Time_Class& b);
};

#ifdef STATIC_AND_INLINE
static
#endif
inline bool operator<(const Time_Class& a,const Time_Class& b)
{
        return (a.the_time < b.the_time);
}
#ifdef STATIC_AND_INLINE
static
#endif
inline bool operator>(const Time_Class& a,const Time_Class& b)
{
        return (a.the_time > b.the_time);
}


#endif
```

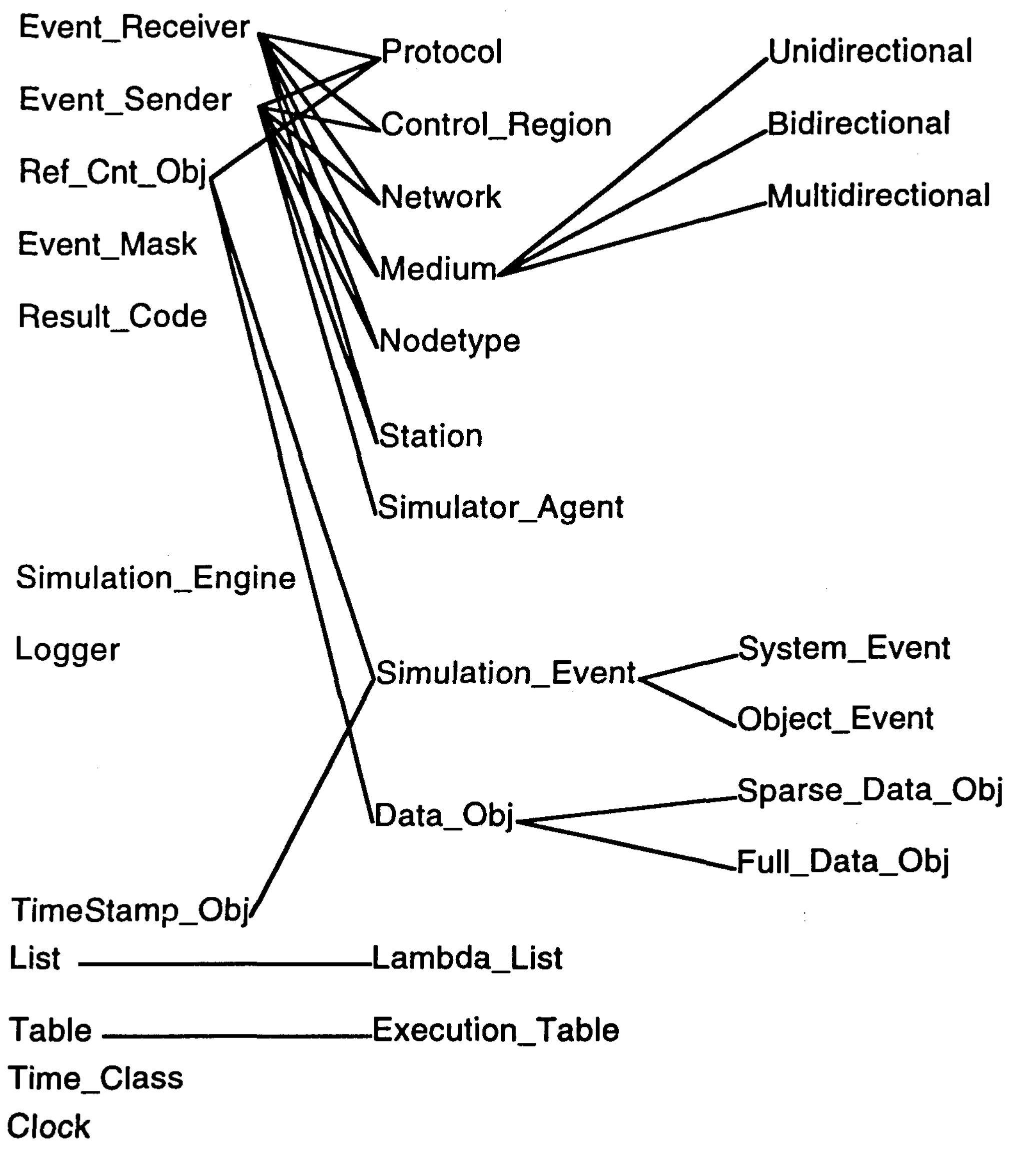
```
// config.h - presently empty
// pairings.h - not yet implemented

#ifndef PAIRINGS_H
#define PAIRINGS_H

#include <pair.h>
#include <bool.h>
#include <stdlib.h>
#include <assert.h>
#include <stdio.h>
#include <map.h>

template <class T, class U>
class pairings {
  private:

  protected:
        map< U , T , less<U> > it; // we may want to change this later - I hope not

  public:
        pairings(void);
        T& lookup(U);
        bool add(T&,U);
        bool remove(U);
};


template <class T,class U>
bool pairings<T,U>::add(T& what,U key)
{
        it[key] = what;
        // I should put in the error checking and handling that I was
        // planning on, but I doubt anyone will use it
        // and I don't know how to do it using the STL
        return true;
}

template <class T,class U>
pairings<T,U>::pairings(void) {
// a do nothing constructor
}

template <class T,class U>
T& pairings<T,U>::lookup(U key)
{
        return it[key];
}

template <class T,class U>
bool pairings<T,U>::remove(U key)
{
        it.erase(key);
        // I should put in the error checking and handling that I was
        // planning on, but I doubt anyone will use it
        // and I don't know how to do it using the STL
        return true;
}
#endif
```

```
/*
        Run-Time Sumulation Library test simulation agent and station class
        (c) 1996 Andrew Chen

        No guarrantees.

        main.c for

*/

#ifndef SLASH_CAN_BE_IN_FILENAMES

#include "../rtsl/config.h"
#include "demo_Sim_Ag.h"
#include "../rtsl/RTSLUtils.h"
#include "../rtsl/Globals.h"

#else

#include "config.h"
#include "demo_Sim_Ag.h"
#include "RTSLTUtils.h"
#include "Globals.h"

#define NEED_CONSOLE
#include <console.h>

#endif

int main(int argc,char** argv)
{
#ifdef NEED_CONSOLE
        argc = ccommand(&argv);
#endif

        if (argc < 2) die("Need a simulation file.\n");

        gSimulation_Agent = new demo_Sim_Ag(argv[1]);
        gSimulation_Agent->Do_Simulation();
}
```

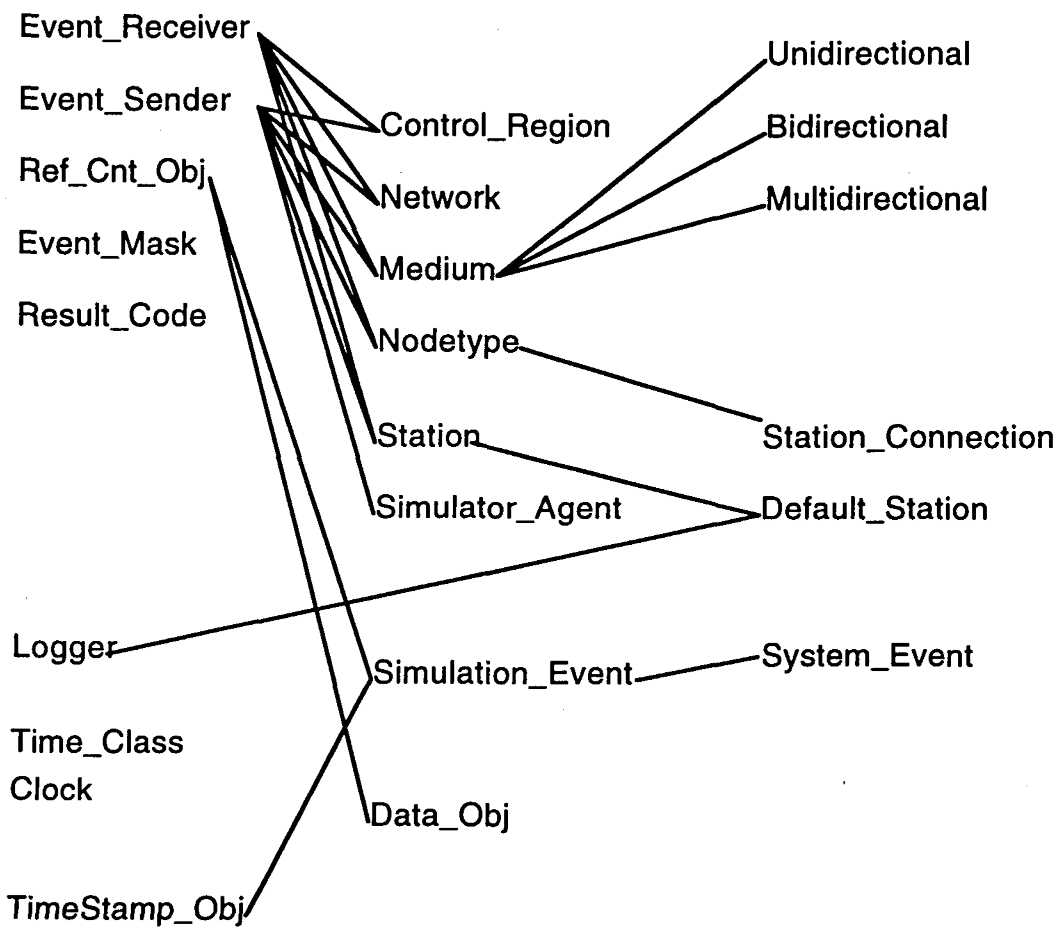
```
#include "demo_Sim_Ag.h"
#include "../rtsl/Globals.h"

#include "../rtsl/Station.h"
// that was included only for this hacked demo
#include "../rtsl/System_Event.h"
// as was that
#include <iostream.h>
// and that
#include "../rtsl/Default_Station.h"
// and that
#include "../rtsl/Station_Connection_Echo.h"
//and that
#include "../rtsl/Station2Station_Connection.h"
// and that

Result_Code* demo_Sim_Ag::Handle_Event(System_Event* the_Event,Station* a_Station)
{
// this function is just a quick hack for the demo prototype
// ideally it should queue the events based on a timestamp into the event queue
// but here it just send them as soon as they come in

// also, ideally that would be a Simulation Event * as an argument and
// not a System Event *
        return (a_Station->Handle_Event(the_Event));

}

bool demo_Sim_Ag::Process_Event(System_Event* the_Event)
{
// should process the event - unimplemented in reality right now.
// the code here is just for the demo.

        if (the_Event) {
                switch (the_Event->thy_Event_Code()) {
                  case new_station:
                        the_Station = new Default_Station(cout);
                        return 1;
                        break;
                  case send_most_recently_created_station_test_event:
                        the_Event = new System_Event(send_self_test_event);
                        assert(the_Station);
                        the_Station->Handle_Event(the_Event);
                        the_Event->Destroy();
                        return 1;
                        break;

                        // the following is just a hack
                        // that's why the reference counts on these are probably off

                  case send_data_event:
                        gSimulation_Agent->Send_Event(
                                new System_Event(send_data_event,
                                        new Sim_Packet),
                                the_Station);
                        return 1;
                        break;

                        // same here

                  case new_echo_node:
                        Station_Connection_Echo * TSCE;
                        TSCE = new Station_Connection_Echo;
                        TSCE->Attach_to(the_Station);
                        gSimulation_Agent->Send_Event(
                                new System_Event(new_connection_node_event,TSCE),
                                the_Station
                        );
                        return 1;
                        break;

                        // and here

                  case new_s2s_conn_node:
                        Station2Station_Connection * S2SC;
                        S2SC = new Station2Station_Connection;
                        S2SC->Attach_to(the_Station);

                        gSimulation_Agent->Send_Event(
                                new System_Event(new_connection_node_event,TSCE),
                                the_Station
                        );
                        return 1;
                        break;

                  default:
                        assert(the_Station); // just a hack right now...
                        the_Station->Handle_Event(the_Event);
                        the_Event->Destroy();
                        //return 1;
                        return 0;
                        break;
                };
        } else {
                switch (state) {
                        case 0: the_Station = new Default_Station(cout);
                                state = 1;
                                return 1;
                                break;
                        case 1:
                                the_Event = new System_Event(send_self_test_event);
                                the_Station->Handle_Event((Simulation_Event*)the_Event
);

                                the_Event->Destroy();
                                state = 2;
                                return 1;
                                break;
                        case 2: return 0;
                                break;
                                default: return 0;
                                break;
                }
        }
}

static char * SSTE = "Send Self Test Event";
static char * SSTP = "Send Self Test Packet";
static int one = 1;

demo_Sim_Ag::demo_Sim_Ag(char * thefile) : Simulator_Agent(thefile)
{
//      delete my_event_queue;
/*************************************************************************
*
                                                                        *
*               my_event_queue = new Event_Queue;
                                                    *
*               // sacrifice the read in EDF file so we can    put    our    own
events on there,   *
*               // the arg-based ones that ReadEDF presently doesn't know how to handl
e.   *
*               my_event_queue->push(new System_Event(new_station));
                    *
*
                                                                              *
*               my_event_queue->push(new System_Event(make_pair(one,&SSTP)));
           *
*
                                                                              *
************************************************************************************/
        // the above commented out because we actually do want thd old-style EDF
        // for now
}

Nodetype * demo_Sim_Ag::Name2Node(char* the_node_name)
{
        cerr << "demo_Sim_Ag::Name2Node not implemented yet" << endl;
        cerr << "could not look up node name: " << the_node_name << endl;
        die("Internal error - unimplemented function demo_Sim_Ag::Name2Node"
                " was called.\n");
        return 0;
}
```

```
#ifndef DEMO_SIM_AG_H

#define DEMO_SIM_AG_H
#include "../rtsl/Simulator_Agent.h"

#include "../rtsl/Station.h"

#include "../rtsl/Nodetype.h"

class demo_Sim_Ag : public Simulator_Agent
{
  protected:
        int state; // this is only here as a quick hack right now
        Station* the_Station; // as is this

  public:
        bool Process_Event(System_Event*);
        Result_Code* Handle_Event(System_Event*,Station*); // this is just a quick hac
k
        demo_Sim_Ag(char*);
        Nodetype * Name2Node(char*);
};

#endif
```

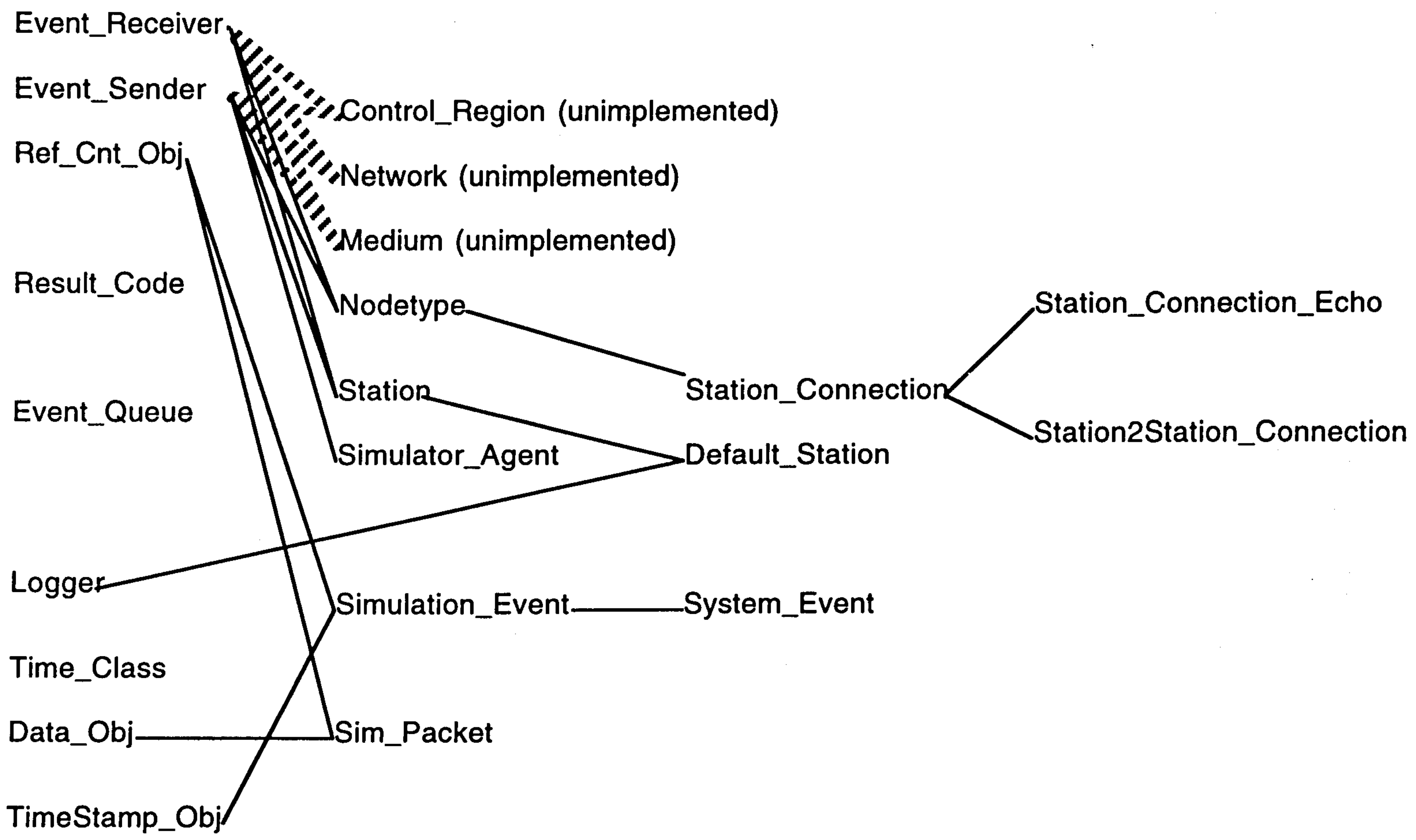
## Appendix B1: Original Class Hierarchy



## Appendix B2: Intermediate Class Hierarchy



## Appendix B3: Latest Class Hierarchy



## Appendix C1: Introduction to Prolog

Prolog:

line 1 <u>a:-b</u>.

line 2 <u>b</u>.

These are prolog statements.

<u>a:-b</u>.

is a "rule".

"a" is the left-hand side (or precedent),

"b" is the right-hand side (or antecedent).

<u>b</u>.

is an "axiom".

After those prolog statements are entered,
a user might be dealing with the prolog front end and might ask:

<u>b</u>.

This "matches" with the axiom in line 2,
so this would "return" "true".
The user might ask

<u>a</u>.

This "matches" with the precedent of the rule in line 1,
so now the prolog interpretter tries to "mathc" the right hand side of line 1.
The right hand side of line 1 is "b".
This is matched by the axiom <u>b</u>. in line 2,
so the result would be "true".
Another user might ask

<u>c</u>.

This doesn't match with anything,
so the result of that query would be "false".

Prolog rules can have variables,
which always begin with a capital letter.

line 3 <u>or(X,Y):-X</u>.

line 4 <u>or(X,Y):-Y</u>.

line 5 <u>and(X,Y):-X,Y</u>.

line 6 <u>istrue(X):-X</u>.

A user might query

<u>istrue(b)</u>.

which would match with line 6,
and so "X" would become *bound* to "b".
Line 6 would then be interpretted as an intention to try to match "b".
"b" would be matched on line 2, so the query would return "true".

Another user might query

<u>or(c,d)</u>.

this would match with line 3 and try to satisfy "c"
(with "X" bound to "c") in this case.
This would fail, so line four would be attempted to be matched.
This would try to satisfy "d" (with "Y" bound to "d").
This too would fail,
so "or(c,d)" would evaluate to "false".

Another user might query

<u>and(a,or(b,c))</u>.

To show what would happen for this query,
we will adopt a trace notation.

```
goal          and(a,or(b,c)).
match line 5  subgoals: a, or(b,c)
        goal  a
        match line 1  subgoal: b.
                match line 2.
                true.
        goal  or(b,c).
        match line 3  subgoal: b.
                match line 2.
                true.
        true.
true.
```

We've just about implemented the normal boolean logic predicates,

(which, as you can see, are practically built into Prolog).
We're just missing "not".
To implement "not" though, we need to mention several more things.
First amoung those is the matter of bound and unbound variables.
An example of bound and unbound variables is the following:

line 7 parent(bob,jane).
line 8 parent(bob,jim).

A user might query

parent(X,Y).

And the query would match line 7 first,
and the "X" would become bound to "bob",
and the "Y" would become bound to "jane".
Then it would return "true" with X = "bob" and Y = "jane",
and it would ask the user if another solution should be searched for.
If the user says no, the query stops.
If the user says yes, then the next line is matched (line 8)
and "X" is bound to "bob" and "Y" is bound to "jim".
Again the user would be told this and asked if another result is desired.
Presently there is no other result,
so the query would stop regardless of what the user answered.

Then there is the "=" operator.

line 9 assignOrEqual(X,Y):-X=Y.

So if a user queried:

assignOrEqual(X,a).

The result would be (with "a" bound to "Y") an "X = a. Continue?"
where the "Continue?" is the asking
if we want to look for another solution.
Likewise, if a user queried:

assignOrEqual(a,b).

The result would be "true" because the "a:-b" rule would be used.
So if a user queried:

assignOrEqual(a,a).

The result would be "true" because a = a.
If a user queried:

```
assignOrEqual(a,c).
```

the prolog interpretter would return "false"
because there is no rule that it can match to either a or c
to get them to be the same.
If a user queried:

```
assignOrEqual(X,Y).
```

the prolog interpretter would try all the axioms that it knew about
in order of their entry,
and see which ones would be found equal to each other.
The solutions it would probably present to the user, in order, would be
X = a, Y = a
X = a, Y = b
X = b, Y = a,
X = b, Y = b.

Then there is "cut" or "!".
Cut does not allow backtracking to go to before it.

```
line 10    male(jim).
line 11    isfather1(X,Y):-parent(X,Y),male(Y).
line 12    isfather2(X,Y):-!,parent(X,Y),male(Y).
```

So the query:

```
isfather1(bob,Z).
```

would return "Z = jim"
but ther query:

```
isfather2(bob,Z).
```

would return "false" because the parent rule would match the first one,
and then the male rule would fail,
so the rule would fail and the "!" would prevent backtracking.
Thus the "not" can be represented as:

```
line 13    not(X):-X,!,fail.
line 14    not(X):-.
```

If "X" is true, then the first rule proceeds past the cut,

and “fail”s, returning a “false”.
If “X” is false, the first rule doesn’t match and so the second rule is tried.
There it matches (because it doesn’t require anything,
so it’s true by default), so it returns “true”.

## Appendix C2: Differences between Prolog and the PDL

I. The variations to Prolog that I plan to make in the rule-specifications are the following:

   A. Perl-style “&” in front of all rule-names, everywhere.

   B. Perl-style “$” in front of all variable references, but not in front of variable declarations.

   C. C-style expressions can be used in the clauses.

      1. As in C, zero would be false, non-zero true.

      2. No keeping track of which variables are “bound”.

      3. The expressions would be post-fix.

   D. The rules have precedence over each other in accordance with what would be expected of the object-class hierarchy of protocols and nodetypes.

II. Some examples of the above variations would be as in the following:

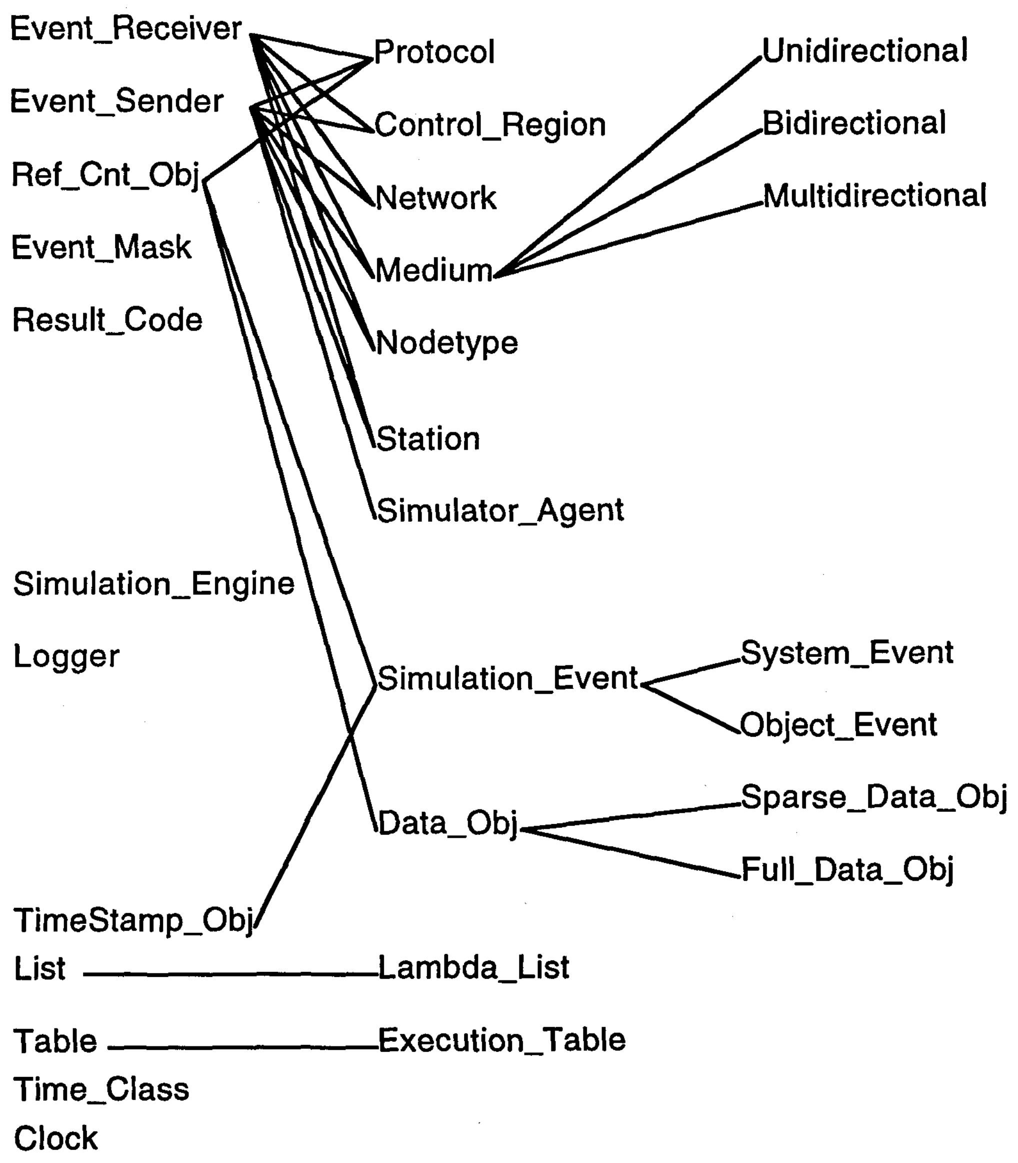
   *A. parent(bob,jim).*
   would instead be
   *&parent(bob,jim):-.*

   *B. add(X,Y,Z):-Z=X+Y.*
   would instead be
   *&add(x,y,z):-$z $x $y + =.*

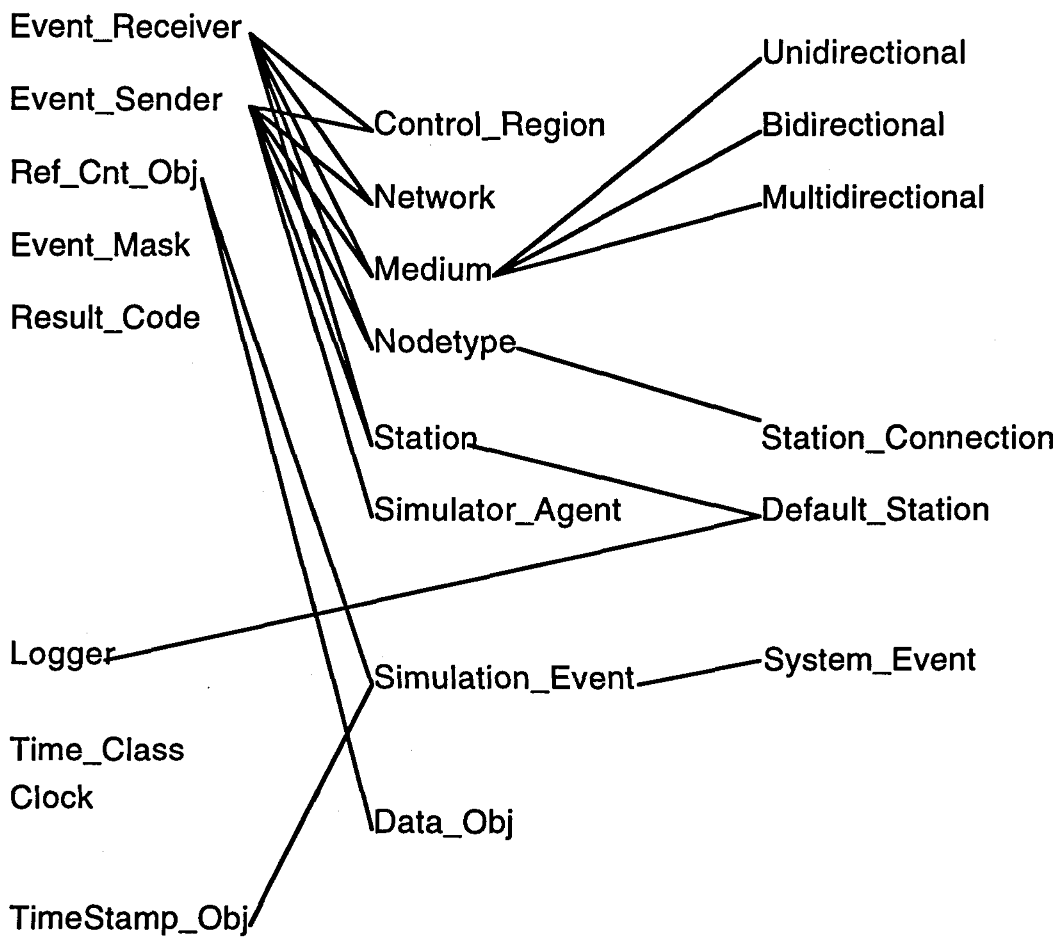
   *C. equal(X,Y):-X=zero,Y=zero.*
   *equal(succ(X),succ(Y)):-equal(X,Y).*
   would instead be
   *&equal(x,y):-$x $y ==.*

   D. The functionality of the Prolog “add” in example B would require that it be split in two - one for the case where Z is bound and “add” is just testing the validity of the statement - the other for the case where Z is not bound and the “add” rule is binding a value to Z. The later is the one actually specified above. The former would be done via the following:
   *&check_sum_equal(x,y,z):-$z $x $y + ==.*

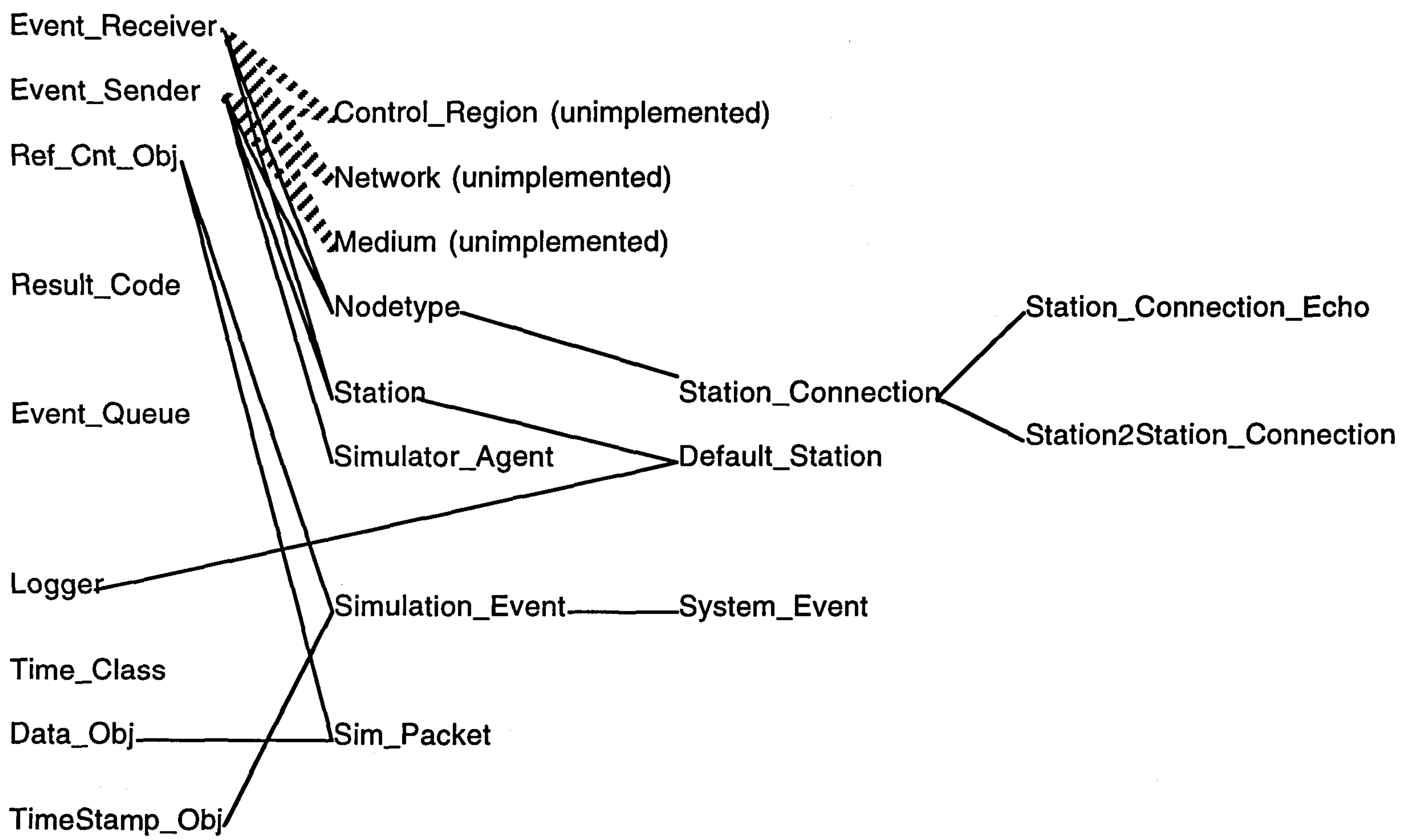
## Appendix B1: Original Class Hierarchy



## Appendix B2: Intermediate Class Hierarchy



## Appendix B3: Latest Class Hierarchy



## Appendix C1: Introduction to Prolog

Prolog:

line 1 a:-b.

line 2 b.

These are prolog statements.

a:-b.

is a “rule”.

“a” is the left-hand side (or precedent),

“b” is the right-hand side (or antecedent).

b.

is an “axiom”.

After those prolog statements are entered,
a user might be dealing with the prolog front end and might ask:

b.

This “matches” with the axiom in line 2,
so this would “return” “true”.
The user might ask

a.

This “matches” with the precedent of the rule in line 1,
so now the prolog interpretter tries to “mathc” the right hand side of line 1.
The right hand side of line 1 is “b”.
This is matched by the axiom b. in line 2,
so the result would be “true”.
Another user might ask

c.

This doesn’t match with anything,
so the result of that query would be “false”.

Prolog rules can have variables,
which always begin with a capital letter.

line 3 or(X,Y):-X.

line 4 or(X,Y):-Y.

line 5 and(X,Y):-X,Y.

line 6 istrue(X):-X.

## Appendix C2: Differences between Prolog and the PDL

I. The variations to Prolog that I plan to make in the rule-specifications are the following:

   A. Perl-style “&” in front of all rule-names, everywhere.

   B. Perl-style “$” in front of all variable references,
   but not in front of variable declarations.

   C. C-style expressions can be used in the clauses.

      1. As in C, zero would be false, non-zero true.

      2. No keeping track of which variables are “bound”.

      3. The expressions would be post-fix.

   D. The rules have precedence over each other in accordance with what would be expected of the object-class hierarchy of protocols and nodetypes.

II. Some examples of the above variations would be as in the following:

   *A. parent(bob,jim).*
   would instead be
   *&parent(bob,jim):-.*

   *B. add(X,Y,Z):-Z=X+Y.*
   would instead be
   *&add(x,y,z):-$z $x $y + =.*

   *C. equal(X,Y):-X=zero,Y=zero.*
   *equal(succ(X),succ(Y)):-equal(X,Y).*
   would instead be
   *&equal(x,y):-$x $y ==.*

   D. The functionality of the Prolog “add” in example B would require that it be split in two - one for the case where Z is bound and “add” is just testing the validity of the statement - the other for the case where Z is not bound and the “add” rule is binding a value to Z. The later is the one actually specified above. The former would be done via the following:
   *&check_sum_equal(x,y,z):-$z $x $y + ==.*