NASA/CR-- 1998 207689

NAG1-1745

IN-04-CR
081603

Air Traffic Network Project Final Report

High-Level Requirement

The high level requirement of the Air Traffic Network (ATN) project is to provide a mechanism for evaluating the impact of router scheduling modifications on a networks efficiency, without implementing the modifications in the live network.

Requirements Breakdown

The high level requirement breaks down into the following lower level requirements:

- *A software simulation must be provide to perform this analysis.*
- *The user must be able to construct multiple network models within the simulation, in order to have a comparison base.*
- *The user must be able to control which scheduling algorithm to use for each network model in the simulation .*
- *The user must be provided with enough feedback to evaluate and compare the impact of the modified scheduling algorithms on the network models.*

Scope of Project Work: 1996-97

The scope of the project was to

- *Provide a proof of concept project on which future work could be based.*
- *Break down the high level requirement. The resulting lower level requirements are listed above.*
- *Design the high level architecture for the software simulation. It must satisfy the requirements described above, as well as be flexible and scaleable enough to provide a foundation for all future work on the project.*
- *Implement an initial simulation which can support* ***simple*** *network models.*
- *Implement an initial user interface to control the simulation and provide* ***simple*** *graphical feedback of a singular simulation statistic.*

High Level Architecture of the Software Simulation

The high level architecture of the software simulation is meant satisfy a combination of good engineering qualities and the student's goals for the learning experience The engineering qualities are:

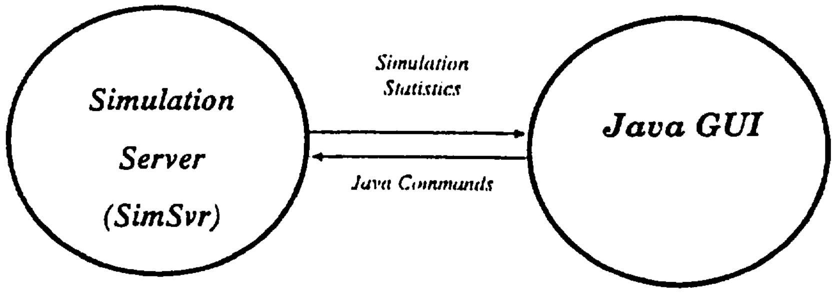
- *Scalability*
- *Platform independence*
- *Modularity*
- *Ease of maintenance*

The student's goals were to :

- *Implement the project using object-oriented methodology.*
- *Learn C++ along the way.*
- *Gain experience with web-related technologies*

The following is a graphical view of the ATN project's high-level architecture.

AIR TRAFFIC NETWORK HIGH-LEVEL ARCHITECTURE



A client-server architecture was implemented between the simulation and the graphical user interface(GUI). This allowed for a loose coupling between the GUI and the simulation, enabling me to build the simulation first, test the simulation in a scalable and incremental manner, and then attached a GUI of my choice later in the process. As a result, there is a defined protocol by which the GUI communicates with the simulation server. The Java GUI issues commands over TCP?IP sockets, which we call "Java Sim£mds", are discussed in Appendix A. All simulation statistics are sent back to the Java GUI, so it can pick & choose the data to watch. This allows this greatest flexibility for growth of the user interface in the future, allowing the decision of what data items to watch to occur within the GUI verses the simulation server..

We decided to implement GUI in Java with the future intention of running it as an applet. This would provide some platform independence, as well as meet the student's goal of gaining experience with web-related technologies (Java). It would also mean wider *availability with access via the Web. Some of the latest technology in simulation is to provide simulation which can be run in a Web browser, thus available to a wider audience. In fact, Java Sim is one of the newest simulation languages supporting this trend of Web-based simulation. Java Sim should be looked into for simulation projects such as this, once it matures and stabilizes.*

We decided to write the simulation server as an object-oriented server in C++ and CSIM18, a C++ based object-oriented simulation language. CSIM18 is more mature than Java Sim and is currently used more by industry. Reference www.mesquite.com for more information on CSIM18. Choosing to write this as object-oriented in C++ aids in the modularity of the project and maintenance of the project by future students.

Initial User Interface Design and Implementation

The user interface requirements for the project was to provide the following capabilities to the user :

- *Ability to load network models into the simulation.*
- *Ability for the user to pick a singular statistics to monitor across the network models as the simulation executes..*
- *Ability to display that statistics in a graphical form that allows for comparison of that statistic between the network models.*
- *Ability to execute the simulation once the network models are loaded.*

The user has the ability to load a maximum of four network models into the simulation via the use of what we term "scenario files". The maximum is set to four because this is the number of different scheduling algorithms identified for this project. Scenario files are just datafiles which contain a collection of Java Simulation commands (see Appendix A) that form a network model. The user interface allows the user to pick scenario files to load into the simulation server.

Once all simulation files are loaded into the simulation server, the user can pick a singular statistic to monitor for a specific router and queue. The available statistics to monitor currently are :

- *Queue service time*
- *Queue utilization*
- *Queue throughput*
- *Queue length*
- *Queue response time*

The statistic chosen is displayed in two graphical forms. One is an instance bar which displays current value of the statistic and the simulation time. The second is a running strip chart of that statistics value as it changes across simulation time.

Once the user has loaded the scenario files and picked the statistic to watch, he can select "Execute" from the simulation menu to tell the simulation server to run the simulation. The simulation will the produce statistical output every five seconds of simulation time. The user will see the instance bar and strip chart being update as the simulation executes.

Appendix A - Java Simulation Commands

All communication with the simulation server, "SimSvr", takes place via "SimCmds". A scenario file consists of the necessary SimCmds to build a single topology in the SimSvr. SimCmds are of the following format:

Java SimCmd <port#> <cmd #> <args for that cmd>

<port #> - service number where the SimSvr is listening. This number must be the same as what the
SimSvr was run with on the command line.

<cmd #> - the following is a list of commands that the SimSvr understands.

0	*- Adds a Node*
1	*- Adds a Link*
2	*- Adds a Connection*
6	*- Sets the simulation time*
10	*- Sets the queue service method to use for a particular topology*
15	*- Execute the simulation*
16	*- Establish a socket for statistical output for a particular topology*
19	*- Prints the content of a particular topology structure*
20	*- Tells the SimSvr to exit*

<args for the cmd>

Arguments for Adding a Node would be:

<net id>	*- index of the network (figure by load order starting at 0)*
<node #>	*- node number (start at 0)*
<AOC|ATC>	*- node type*
<# routers>	*- number of routers this node has.*

For each router list the following as args .

<rtr speed>	*- speed of router n*
<rtr NQ time>	*- time router takes to enqueue items in ms*
<rtr DQ time>	*- time router takes to dequeue items in ms*
<Qsize 1>	*- maximum size of priority queue 0*
<Qsize 2>	*- maximum size of priority queue 1*
<Qsize 3>	*- maximum size of priority queue 2*
<Qsize 4>	*- maximum size of priority queue 3*
<Qsize 5>	*- maximum size of priority queue 4*
<# hosts>	*- number of hosts at this node*

For each host .
<no connections> - number of connections host supports

Arguments for Adding a Link would be:

<net>	*- index of the network (figure by load order starting at 0)*
<link type #>	*- defined as follows.*

0	*Mode-S*
1	*SATCOM*
2	*VHF*
3	*ATM*
4	*T1*
5	*Ethernet*

<src node id>	*- node number representing source node of link*
<dest node id>	*- node number representing destination node of link*
<length>	*- length of the link in KM*
<capacity MB / s>	*- capacity of link in MB / Sec*

Arguments for Adding a Connection would be:

<net_id>	*- index of the network (figure by load order starting at 0)*
<#hops>	*- number of nodes the connection traverses, including source & destination*
<src host id>	*- node number representing the connections start point*

For each node the connection travels through list .

<hop id 1>	*- node number*
<rtr id 1>	*- index of router (figure by load order starting at 0)*

<dest host id> - node number representing the connections end point

<message type> *these 2 are defined as follows:*
<priority>

0	*Urgent Communications*
1	*Flight Safety Messages*
2	*Weather Broadcast*
3	*Flight Regularity Messages*
4	*Aeronautical Info Service Messages*

<distribution>	*- distribution to use*
<mean>	*- mean arrival rate of packets (double)*
<range>	*- mean deviation (double)*
<start>	*- simulation time at which to start the connection*
<duration>	*- number of simulation clock ticks the connection should last*

Arguments for setting the simulation time:

<time> - double representing how long you want the simulation to run.

Arguments for setting the queue service method for a topology :

<net> - index of the network (figure by load order starting at 0)
<method #> - defined as follows

0	*- service queues uniformly in sequence*
1	*- service queues according the priorities*
2	*- service queues according to the queues bandwidth*
3	*- service queues by priority applying bandwidth & aging*

Arguments for executing the simulation:

none

Arguments for establishing output socket :

<net>	*- index of the network (figure by load order starting at 0)*

Arguments for printing content of a network structure:

<net>	*- index of the network (figure by load order starting at 0)*

Arguments for telling the simulation server to exit:

none