Towards a Rigorous Software Architecture Documentation Process: A Demonstration with the Real-time Immersive Network Simulation Environment (RINSE)

Prof. Joseph R. Laracy
Dept. of Systematic Theology
Dept. of Mathematics & Computer Science
Seton Hall University
400 South Orange Avenue
South Orange, NJ 07079
USA
joseph.laracy@shu.edu

Russell Greenspan
CTO
PresenceLearning
530 7th Avenue
Suite 407
New York, NY 10018
USA
russellgreenspan@gmail.com

Abstract
Despite numerous empirical studies and wide-spread, practical experience demonstrating the importance of rigorous documentation in software engineering, many developers continue to treat it as an "after thought." Documentation, particularly of software architecture, should be an integral process of any development group, whether entrepreneurial, academic, or corporate. In this article the authors develop and apply the software engineering principles of Len Bass, Paul Clements, and Rick Kazman for software architecture documentation. The case study involves a relatively large-scale, academic development project aimed at supporting large-scale network security preparedness and training exercises, involving hundreds of players and a modeled network composed of hundreds of networks.

Keywords
Software Engineering, Software Architecture, Software Documentation, Simulation, Information Security

I. Introduction

Motivation

Eoin Woods points out that "as software systems have evolved, so has software architecture, with practices growing to meet each era's new challenges."[1, p. 94]The distinguished software engineers, Len Bass, Paul Clements, and Rick Kazman make a compelling case for the importance of high quality software architecture documentation. They write,

> As we have seen over and over, the software architecture for a system plays a central role in system development and in the organization that produces it. The architecture

> serves as the blueprint for both the system and the project developing it. It defines the work assignments that must be carried out by design and implementation teams and it is the primary carrier of system qualities such as performance, modifiability, and security—none of which can be achieved without a unifying architectural vision. Architecture is an artifact for early analysis to make sure that the design approach will yield an acceptable system. Moreover, architecture holds the key to post-deployment system understanding, maintenance, and mining efforts. In short, architecture is the conceptual glue that holds every phase of the project together for all of its many stakeholders. Documenting the architecture is the crowning step to crafting it[2, p. 201].

Building on the work of the aforementioned authors and others, in this paper we demonstrate a powerful software architecture documentation technique using the RINSE simulator.

Approach

Much of the information about the RINSE architecture gathered by the authors for this paper was the result of conversations and data sharing with the researchers who developed the system. Our software architecture documentation approach begins with the architectural business cycle. We document important information on the stakeholders, the development organization, the technical environment, and the architects' experience. Next, we examine the salient architecture information such as important classes and in this case, the distributed denial of service attack scenario. After that, we present a number of architectural views, e.g., logical, module, component/connector, deployment, and implementation. Finally, we explore key quality attributes such as performance (e.g., resource demand, resource management, resource arbitration, traditional patterns, and custom patterns), and flexibility/extensibility. In all our work, we attempt to apply and advance the seminal research of Bass et al. [2, Ch. 9].

Software Development Background

RINSE was developed "to support large-scale network security preparedness and training exercises, involving hundreds of players and a modeled network composed of hundreds of LANs."[3, p. 119] The overall goal of the simulator is to present a realistic rendering of network behavior as cyber-attacks are launched and security professionals diagnose incidents and attempt to apply counter measures to maintain network services. Funding for this project came from the US Department of Homeland Security. The team at the University of Illinois successfully built a discrete event simulation system to simulate a terrorist cyber-attack on the nation's critical infrastructure. RINSE allows financial, power, and telecom institutions, both public and private, to participate in "war games" to exercise their systems and command ability.

The high level goal of RINSE is to develop a large-scale real-time network simulation system that is highly extensible. Because of the number of players and duration of the game, hardware redundancy and other fault protection techniques are employed. A variety of novel techniques are employed in implementing RINSE. These include multi-resolution traffic modeling, new routing simulation methods, and a latency absorption technique.

High Level System Description

There are a variety of methodologies available for modeling computer networks, ranging from analytic tools to hardware emulation to simulation. Simulation has the advantage of offering scalability and flexibility. Obviously, in a security scenario like cyber-attacks, it is preferable to attack a simulated network rather than a real one.

RINSE has a variety of capabilities. Some capabilities which are common to other systems in this domain area are parallel execution, discrete event models, and real-time support. RINSE is different from similar systems in its employment of multi-resolution traffic models. These models facilitate human/machine real-time interaction as well as increase efficiency for various attack schemes.

RINSE can be divided into five major modules:

1) iSSFNet network simulator
2) Simulator Database Manager
3) SQL Database
4) Data Server
5) Network Viewer (clients)

iSSFNet Network Simulator

iSSFNet, formerly called DaSSFNet is a robust network simulator. It relies on the common API for parallel simulation of networks, the Scalable Simulation Framework (SSF). The iSSF kernel, which in many ways follows the standard kernel pattern handles support functions and synchronization for iSSFNet. A summary of important classes is provided in a later section.iSSFNet currently runs on the NCSA cluster. These parallel machines allow large networks to be simulated in real-time. Distributed execution is supported by a composite synchronous/asynchronous conservative synchronization mechanism.

Simulator Database Manager

The database management system connects directly to each machine in the simulator. It transmits data from iSSFNet to the SQL database as well as control signals from the database to the simulator.

Data Server

Client applications such as Network Viewer interact with the simulation through this component. The Data Server provides monitoring and control capabilities to system administrators playing the game. It also supports authentication for users, enables Network Viewer to access recent changes to the database via XML-based remote procedure calls, and transmits DML information about client networks.

Network Viewer

This Java based client application enables users to view their network during a scenario. When Network Viewer polls the Data Server for data, the Data Server reads the database and returns the requested information. “Super users” exist which manage the game and view the client activity. They can use Network Viewer in addition to their special tool set which enables them to inject surprises into the game.

A command prompt is also provided in Network Viewer which supports five types of commands.

1. Attack - Super users initiate denial of service attacks with this command. The DDoS scenario discussed later is an example.
2. Defense - System administrators may filter packets and employ other techniques to protect their network.
3. Device Control - System administrators may reboot or disable network devices such as routers.
4. Diagnostics - System administrators may assess their network health.
5. Simulator Data - Super users may control the simulator output or monitor a specific element of the networks in the game.

Relevant Research

Much research has been conducted documenting the common themes and architectural patterns within simulation modeling software. We used this research as a foundation to know what to look for and what to expect in the RINSE architecture. As Gustavson et al. explain, most of the patterns used are structure-oriented, allowing the ability and desire to create flexible and reusable classes within the problem space[4].Neu and Russ explain that organizations often face the same problems when designing their simulations. Because of this, static process models can function as templates, later being molded into actual process descriptions. Architectural patterns can help, such as “how to model or implement typical situations, e.g., how to handle resources, defects, etc.”[5, p. 3] To achieve reuse, the authors suggest modularizing not only the code base, but also the models. Klein et al. describe an interesting distinction between “static” and “dynamic” model elements. Static elements do not change as the simulation executes, while dynamic elements have the potential to change. Dynamic input parameters are received by the simulation at runtime from the sources involved in the simulation[6].

II.Architectural Business Cycle (ABC)

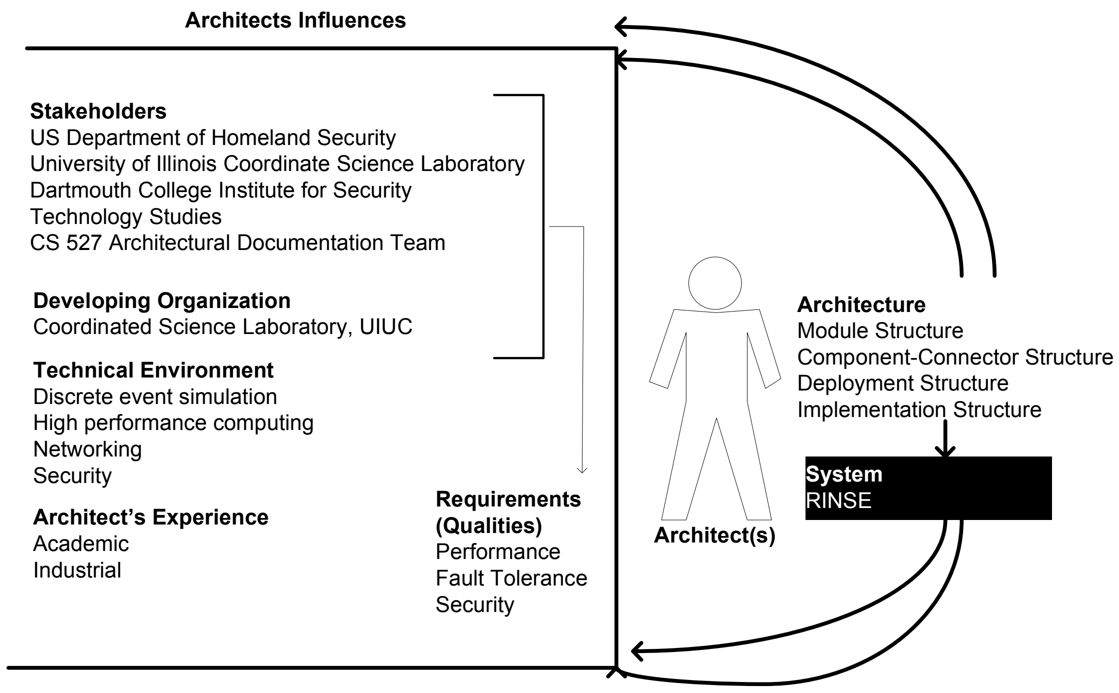

Architects Influences
Stakeholders
US Department of Homeland Security
University of Illinois Coordinate Science Laboratory
Dartmouth College Institute for Security Technology Studies
CS 527 Architectural Documentation Team
Developing Organization
Coordinated Science Laboratory, UIUC
Technical Environment
Discrete event simulation
High performance computing
Networking
Security
Architect's Experience
Academic
Industrial
Requirements (Qualities)
Performance
Fault Tolerance
Security
Architect(s)
Architecture
Module Structure
Component-Connector Structure
Deployment Structure
Implementation Structure
System
RINSE

FIGURE 1 Architectural Business Cycle Diagram

Stakeholders

RINSE is a continuation of projects funded by Defense Advanced Research Projects Agency (DARPA) and the National Science Foundation (NSF). It was also sponsored by the US Department of Homeland Security. The principal development organization is the University of Illinois Coordinate Science Laboratory. The Dartmouth College Institute for Security Technology Studies also contributed. The Architectural Documentation Team is a stakeholder as well.

Development Organization

The Coordinated Science Laboratory (CSL) team was led by Professor David Nicol. Nicol held the following appointments at the University:

- Professor, Department of Electrical and Computer Engineering
- Affiliate Professor, Department of Computer Science
- Research Professor, Coordinated Science Laboratory
- Chair of the Computer Engineering Group, ECE department

His research interests are in the following areas:

- High Performance Computing
- Modeling and Simulation of Large-Scale Systems
- Networks
- Cyber-security

Professor Nicol was assisted by Dr. Michael Liljenstam, Post-Doctoral Research Associate in the Center for Reliable and High-Performance Computing. Dr Liljenstam's areas of expertise are multi-resolution network modeling and simulation, cybersecurity: modeling internet worms, and inter-domain routing analysis.

Dr. Jason (Xiaowen) Liu, Assistant Professor at the Colorado School of Mines, authored the iSSF kernel while a UIUC graduate student. Graduate student Yougu Yuan architected the framework of iSSFNet based on his experience on other SSF implementations, including the precursor DaSSFNet. His focus was on making it more extensible to enable large-scale network simulation. Graduate student Chris Grier, a security expert, assisted with the development of the client side (JAVA) software. Research Programmer Lara Karbiner, a new member of the team, came onboard to help regulate the development and testing.

Technical Environment

The RINSE development team was comprised of experts in the area of discrete event simulation. Additionally, the UIUC team enjoyed expertise in high performance computing, networking, security, and wireless systems. The Dartmouth ISTS group had expertise in security. They developed the models and scenarios for various attacks.

Architects Experience

Professor Nicol's experience:

- 1975-1979 Carleton College , BA (math)
- 1979-1982 Control Data Corporation, programmer/analyst
- 1982-1985 Univ. of Virginia, MS, PhD (computer science)
- 1985-1987 ICASE, Staff Scientist
- 1987-1996 William and Mary, Assoc. Professor of CS
- 1996-2003 Dartmouth, Professor of CS and Chair
- 2002-2003 Assoc. Director ISTS
- 2003-2003 Director ISTS
- 2003-present UIUC, Professor of ECE

Professor Nicol's vast experience in network simulation provides the "grand vision" for RINSE. He was assisted in this regard by Dr. Liljenstam, an expert in multi-resolution network modeling and simulation, cybersecurity: modeling internet worms, and inter-domain routing analysis. Mr. Yuan architected RINSE based on his experience with earlier SSFNet systems. He started out developing some packages of the Java SSFNet. While working on a related project, two other students began developing a C++ implementation of SSFNet. The C++ implementation grew in size and complexity and eventually became DaSSFNet. However, the DaSSFNet APIs lacked extensibility for the new features they hoped to include. As a result, iSSFNet was built from scratch using the lessons learned from DaSSFNet and other past experiences.

III.Architectural Information

Distributed Denial of Service Attack

The following informal use case is intended to give a software engineer trying to learn the RINSE system an understanding of the important classes and their interaction in a particular scenario corresponding to a well-known problem.DDoS, or a distributed denial of service attack is a growing problem for large scale networks, including the internet.In a DDoS attack, the *attacker* seeks to disable a *servervictim* through the exploitation of a computer network.First, the *attacker* identifies a vulnerable server which becomes its *agent*.An example of an *agent* could be an instant messaging server that has the IP addresses of a large number of machines and can easily pass packets to these machines.The attack occurs when the *attacker* signals the *agent* to command all its connected machines, or *zombies*, to bombard the *server victim* with junk traffic.The volume of traffic is so large that the *server victim* can no longer provide its services, whether it is POP3 email services or B2B e-commerce.

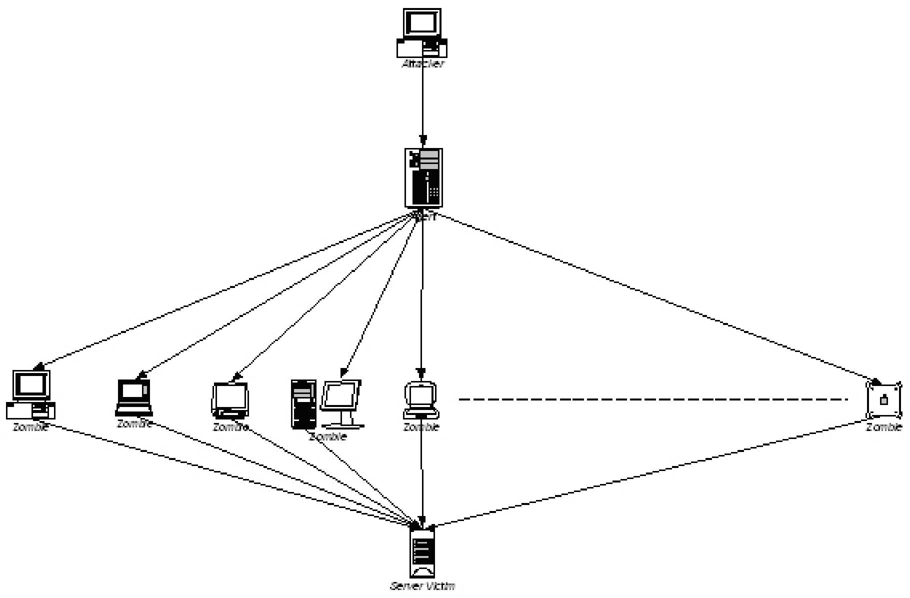


FIGURE 2 Distributed Denial of Service Attack Schematic

DDoS is one of the many scenarios that RINSE hopes to educated system administrators about and prepare their networks against.The following command starts the simulation:

```
ddos_attack attacker server 100 2000
```

ddos_attack – the command
attacker – the *attacker* mentioned above
server – the *server victim* mentioned above
100 – duration of attack in seconds
2000 – rate of bombardment in kilobits/second

The simulation begins when the client sends the command (DDosCmd) and it makes its way to the simulator.It goes through the following classes:

ViewerClient → DataServer → DBMonitor → WebServer

Finally, the command reaches the CmdProxy class and is broadcast to all the physical simulators.There is one Country class associated with each physical simulator.Each Country contains a ResolutionService class which tells the CmdProxy the location of the attacker, or it's Province.Every Country is made up of a connected graph of Province objects.

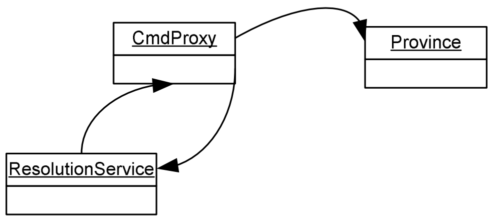


FIGURE 3 Command Proxy Class Diagram

The command is then passed to CmdHandler which communicates with the AuxiliaryOracle and receives a pointer to the *attacker* within the attackerProvince.The Host contains at least one NIC and a protocol graph.The protocol graph contains ProtocolSessions.The ProtocolSession class represents a protocol layer on the ISO/OSI protocol stack. It's the base class for Protocol implementations. The class specifies default mechanisms for how a protocol session should behave. The data path is specified by two methods, push and pop, for receiving data from the protocol session above and below. The data exchanged between protocol layers are encapsulated in a ProtocolMessage object. The exchange of control messages between adjacent protocol layers is implemented through invocations to the control method. Subclasses may override these methods with specific behavior.ProtocolSessions include DDoS, TCP, ICMP, IP, and others.

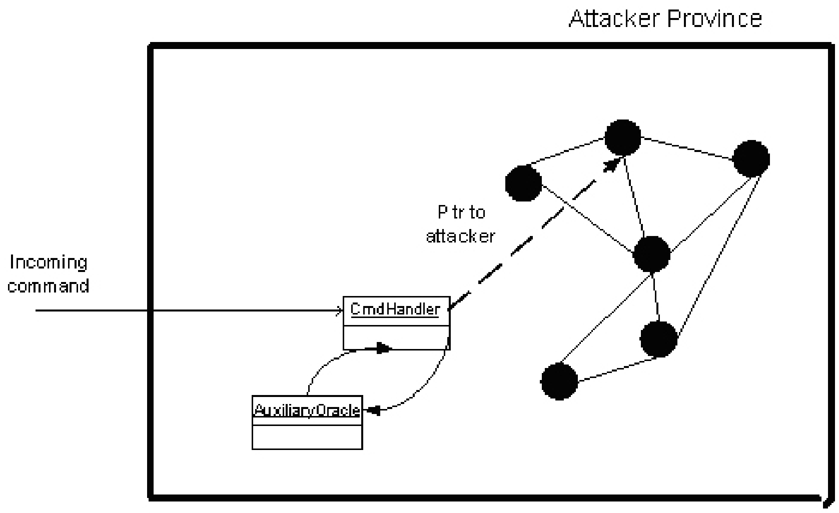


FIGURE 4 Command Handler Class Diagram

Packets are then generated with the following format and sent to the *zombies*.

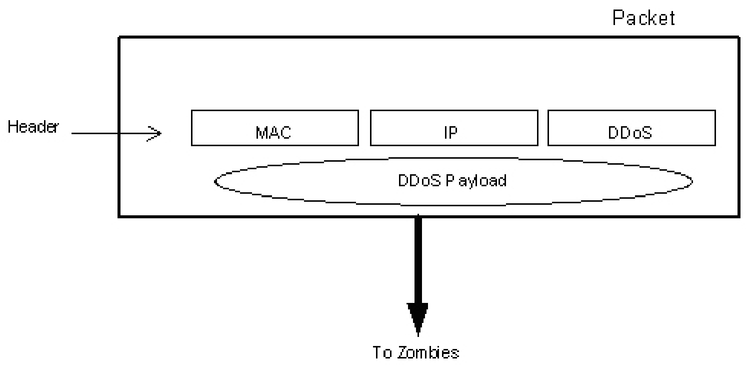


FIGURE 5 Packet Format Diagram

The zombies then assault the *server victim* with the DDoS payload.

While the game is running, clients will periodically poll the database to analyze how their network is performing.Each Province has a ReportManager which reports relevant statistics such as bandwidth usage and processor utilization.Although most information is polled periodically, some major events will alert the client through an event-driven interrupt.Because of the enormous amount of data that will be collected after a week long war game with hundreds of clients, the database cannot store all the information that should be analyzed in the post-game stage.Therefore, a large log file is created to report the results.

Important Classes

Net

The Net class loads the configuration and instantiates the entire model . It reads in the parameters from the passed-in DML branch and configures the hosts, routers, and links. After the configuration, it then goes through the links and connects all the interfaces. Links for subnets are also connected at this point. Once the configuration is finished, it then initializes the traffic, nets, links, hosts and router objects.

Host

The Host class is derived from the ProtocolGraph class and includes support for the IP layer and Network Interface Card (NIC). It instantiates its own timer object that is used at the time of rebooting. In order to register itself, the Host object gets a pointer to the AuxiliaryOracle object via the Province object. It then configures the resources of CPU and memory. For example, it configures the number of instructions that a CPU would need for handling a packet. After configuring the resources, it then configures the protocol sessions and interfaces.

Interface

The Interface class implements the default mac layer. It registers itself with the host-entity and initializes all the protocol sessions. As part of its initialization of the given protocol session, it finds out the class name for this protocol session, creates a new protocol session object, makes it part of the protocol graph, and puts the protocol object in its vector. In case there are no protocol sessions specified for an interface, it would then use the default protocol sessions at both MAC and Physical layer.

Link

The Link class is responsible for resolving the NHI addresses and connecting the NIC to other NICs on the same link. This connection is performed before the actual IP addresses have been assigned to the interfaces.

Monitoring Possibilities

Cisco NetFlow Format

NetFlow data enables extensive near real time network monitoring capabilities. Flow-based analysis techniques may be utilized to visualize traffic patterns associated with individual routers and switches as well as on a network-wide basis (providing aggregate traffic or application based views) to provide proactive problem detection, efficient troubleshooting, and rapid problem resolution.

TCP Dump Format

It provides support for dumps of packet data in tcpdump format. The Instrumentation class is used to monitor the internal state of one TCP session at one end of the connection and to write data to a dumpfile.

Domain Modeling Language

DML is a language used to describe networks to be simulated in RINSE. Networks are described as combinations (layers) of other networks or as a collection of devices such as routers, switches, hosts, etc. Network administrators can describe their network topology using this language and submit the DML to the RINSE team for integration into the game. DML is similar to XML in that it has a simple syntax consisting of nested name-value pairs. In addition, DML has features that allow nodes within the hierarchy to act as pointers to other nodes in the hierarchy. For example, using the keyword "_extends", many nodes can point to one common node.

To construct a Network in RINSE, DML is used as follows:

```
#=====================
# Network model
#=====================
Net [
# support interaction
support_interaction true

# for pseudo dns
dns_entry [ name alpha.aaa.com nhi 2 ]
dns_entry [ name beta.ccc.edunhi 0:0 ]

# subnet 0
Net [
id 0
_extends .dict.basicNet# use the entry in the dictionary
]

# hosts in this level of Net
host [
id 2
# get the graph data from the dictionary section
graph [ _extends .dict.host_graph ]
interface [ id 0 _extends .dict.iface ]
]

# links in this level of Net
link [ attach 2(0) attach 0:0(0) delay 0.2 ]
]
```

FIGURE6 Domain Modeling Language Example

For more information on DML, visit: http://www.ssfnet.org/SSFdocs/dmlReference.html

IV.Architectural Views

As Bass, et. all, explain, views are probably the best way to convey a software's architecture. Views capture a structure, and "documenting an architecture is a matter of documenting the relevant views." [9] In essence, through graphical and annotative descriptions, views provide cross-sections into thefundamentals of the software's architecture.

Logical view

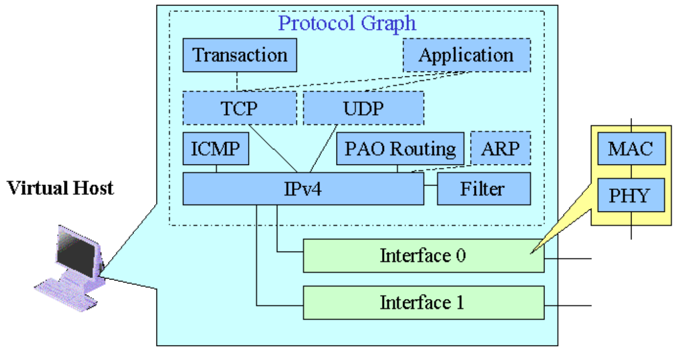
NETWORK-LEVEL CLASSES

iSSFNet tries to only leak a minimum of simulation details and present the normal users basic units to build a network. The following classes should be conceptually familiar to most network researchers:

Net

This class represents a network. It is composed of some smaller nets (subnets), hosts/routers, and links.

Host



Source: http://www.cs.dartmouth.edu/~yuanyg/iSSFNet/doc/imgs/vhost.gif

FIGURE 7 Protocol Graph Diagram

Both hosts and routers are represented by this class. Each host is a ProtocolGraph with one or more Interfaces.

Protocol Graph

Each protocol graph is composed of several ProtocolSessions such as IP, ICMP, etc.

Interface

This is the representation of a network interface card. It contains the MAC layer and physical layer protocol sessions.

ProtocolSession

Each protocol session is a model of one particular protocol, e.g., TCP, IP, ICMP, etc.

Link

This represents a physical link that connects multiple network interface cards.

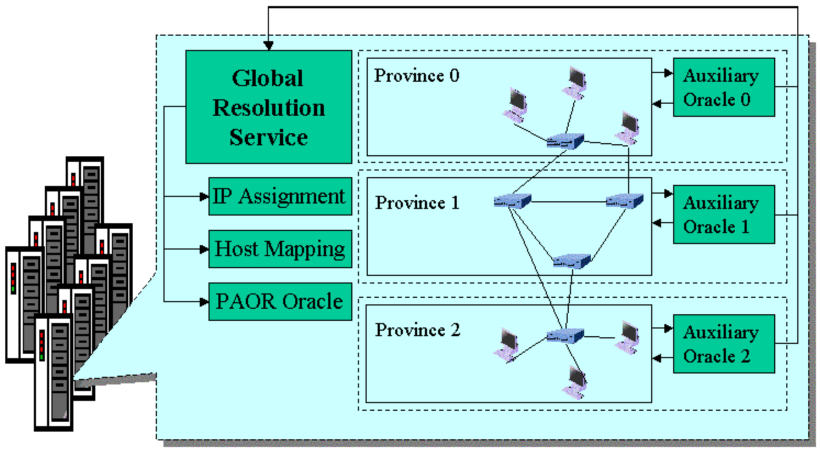
ProtocolMessage

Each protocol implements its own protocol header. It should be an extension of this base class.

Packet

This is the smallest data unit that is sent over a link. Usually it is composed by several ProtocolMessages, just the same as in the real life.

SIMULATION-LEVEL CLASSES



Source:http://www.cs.dartmouth.edu/~yuanyg/iSSFNet/doc/imgs/infrastructure.gif

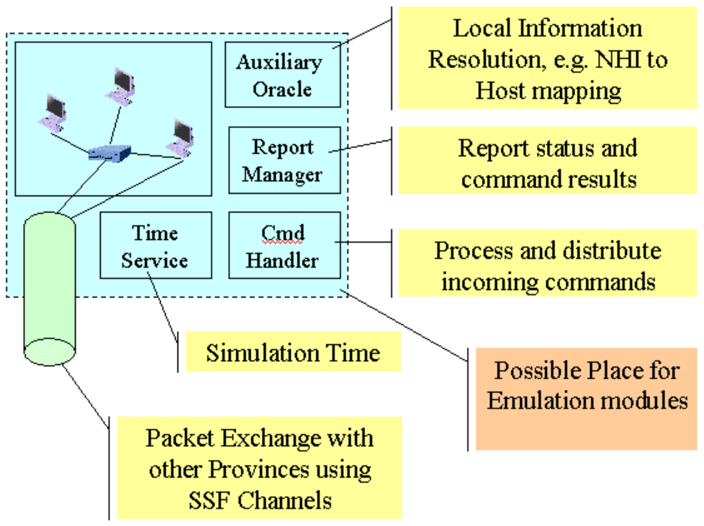
FIGURE 8 Infrastructure Diagram

As a simulator, classes other than the network-level ones are needed to glue them together and provide essential simulated environment information. The main classes that accomplish this task are as follows:

Country

This is literally the main controller of the simulation. It instantiates HostEntities for each physical machine that runs the simulation. It also instantiates ResolutionService that resolves global information such as NHI to IP address mapping.

Province


Auxiliary Oracle
Report Manager
Time Service
Cmd Handler
Local Information Resolution, e.g. NHI to Host mapping
Report status and command results
Process and distribute incoming commands
Simulation Time
Possible Place for Emulation modules
Packet Exchange with other Provinces using SSF Channels

Source:http://www.cs.dartmouth.edu/~yuanyg/iSSFNet/doc/imgs/province.gif

FIGURE 9 Province Diagram

Province is a container of hosts/routers. In iSSFNet design, it is equivalent to timeline or alignment if the users are more familiar with the other terms. Each simulation may have one or more Province instances. Each of them runs in its own timeline, which means in any given moment, the simulated clock in different HostEntities won't necessarily be the same. A Province sits in the background and provides services such as current (simulated) time to the hosts inside, it also provides means to distribute packets to hosts/routers in the other Province instances. Moreover, it processes/distributes external user command received by the CmdProxy to allow user interactions with a host, an interface, or a protocol session in the simulator.

ResolutionService

Hosts or routers or a protocol session sometimes need to know information that is only available globally. For example, an interface may want to get its own MAC/IP address assigned given its NHI address, the user may want to know which Province contains a specific host, given the host name, etc. such information is provided by the resolution service. In many cases, a host may query such information indirectly using the AuxiliaryOracle.

AuxiliaryOracle

Each Province has its own AuxiliaryOracle. It can forward the queries on global information to the ResolutionService, and it also provides local detail information within this Province. For example, given a host NHI address, it can return the host pointer. The local information can only be obtained by hosts or other objects within this Province.

CmdProxy

It opens sockets and process user commands, distribute the commands to the appropriate Province, which further distribute the commands to Host, then to Interface or ProtocolSession. The results of the commands go back using a similar path and are sent out by the CmdProxy. With some change in the MACRO (remove -DUSE_CMD_SOCKET in the Makefile), it should be able receive input from the keyboard instead of the socket.

Module View

SSF VIEW

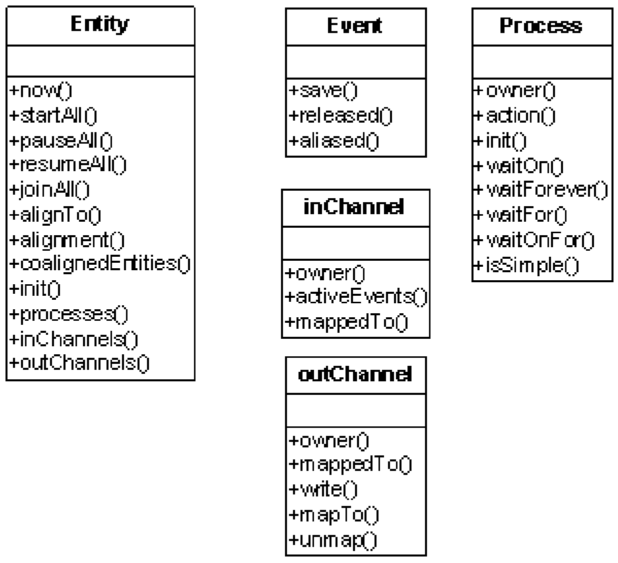


FIGURE 10 SSF View Classes

Entity

Entity serves as the base class for all simulation components, such as hosts, routers, links, and TCP sessions. It provides a container mechanism for defining alignment relations among a model's pieces. All such co-aligned entities interact through event exchange on channels which is taken into account by the underlying simulator at the time ofmapping these entities to the corresponding processors.

Event

Event is the base class for the quantum of information flowing over channels, such as the protocol packets and timers.

Process

Process is the base class for describing an entity's behavior, such as the run-time behavior of protocols. Each instance of a process is associated with an entity. This instance may wait for input arriving on inchannels or wait for time to elapse. In addition, it may also wait on channels of entities co-aligned with its owner.

inChannel, outChannel

These classes serve as the communication endpoints for event exchange, such as the protocol interaction. Each instance of these classes belongs to a specific entity. They provide multicase in-out and bus-style channel mappings. However, the outchannels have a transmission delay associated with them.

LAYERED VIEW

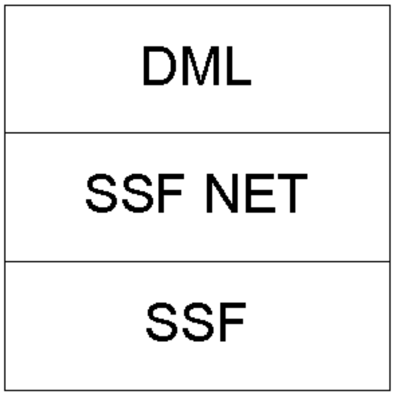


FIGURE 11 Layer Diagram

The DML layer is responsible for the network configuration. It is a framework in itself that accesses the configuration file semantics and its structure. Thus, it essentially acts as a data source for the network models to be simulated. In a network environment, only the central server needs to host this configuration file. All the clients simply retrieve their network model data from this one location.

The SSFNet layer serves as the core layer that knows what objects to instantiate at what point based on the data from the above-mentioned DML layer. It is responsible for the simulation of domain internetworking that is accomplishedvia the important classes mentioned later in this document.The SSF layer resides at the bottom of this layered architecture providing a generic simulation framework that is applicable for almost all sorts of simulation. It provides some fundamental objects from which specific simulations (such as iSSF Net itself) need to derive classes to implement their corresponding functionalities.

Component/Connector view

MAIN VIEW

The following sequence diagram shows the interaction of objects at the time the main method gets called during the load up of iSSF Net. At the time of startup, data is retrieved via the dmlConfig object. From this point on, the core objects of Country and Net use this pointer back to this DML configuration file to instantiate themselves and their corresponding objects:

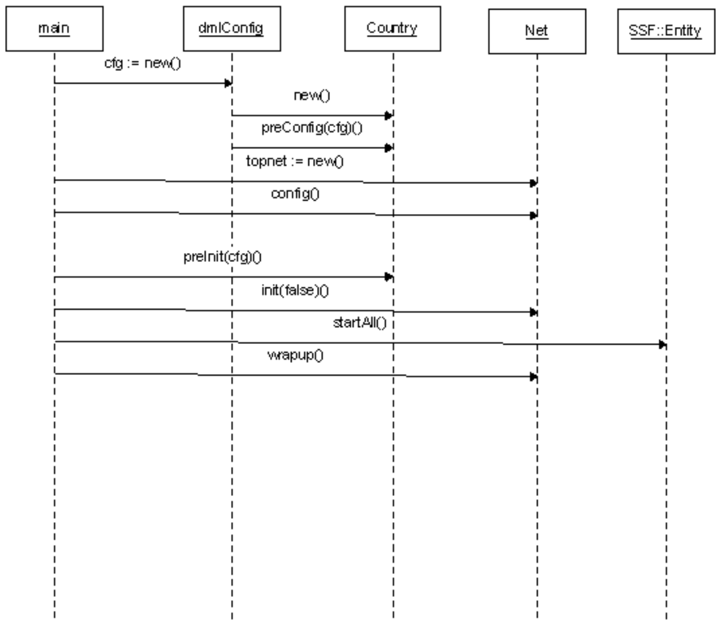


FIGURE 12 iSSF Net Sequence Diagram

NET VIEW

The Net object is solely responsible for loading the whole network model and the associated objects as depicted below. This loading requires the creation of the necessary network-support objects of routers, interfaces, and links:

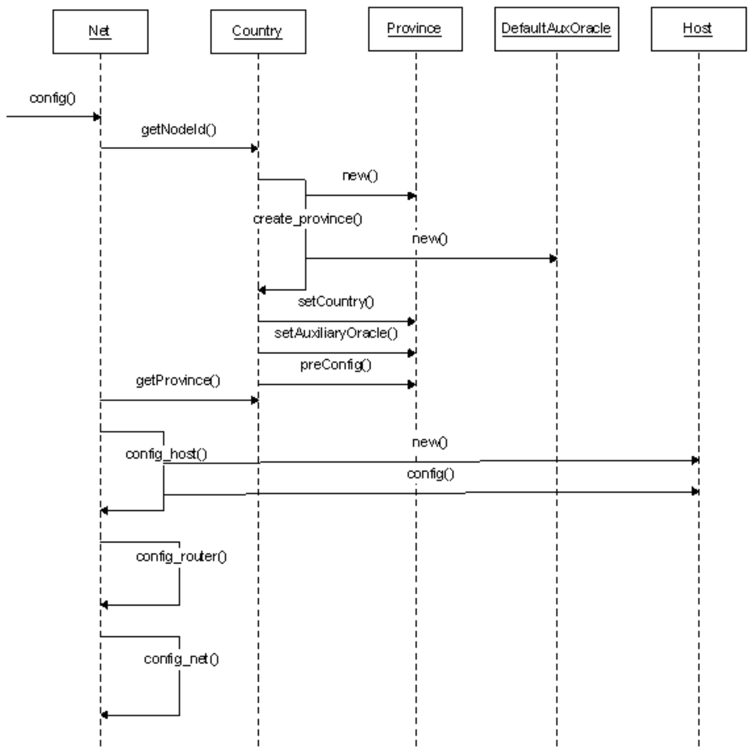


FIGURE 13 Net Sequence Diagram

NETWORK VIEWER/DATASERVER CONNECTOR VIEW

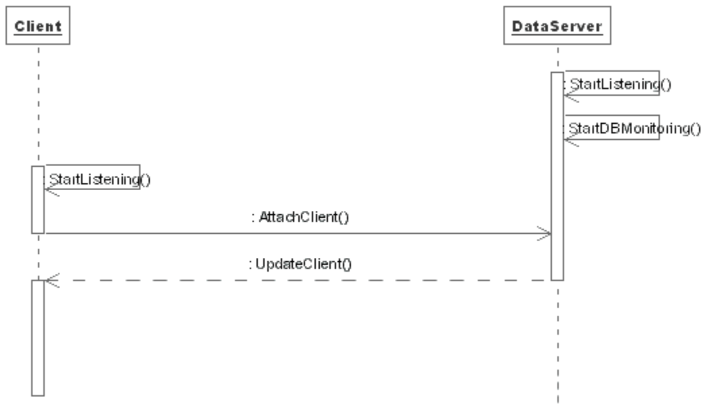


FIGURE 14 Network Viewer / DataServer Connector Diagram

The DataServer calls StartListening() and StartDBMonitoring() during initialization. At this point, the DataServer is ready for Clients to connect. Clients then call StartListening() to begin listening for messages from the server, then attach themselves to the DataServer via AttachClient(). The Dataserver keeps track of all connected Clients, and as needed calls UpdateClient() on each Client via RPC, passing whatever data needs to be sent.

Deployment View

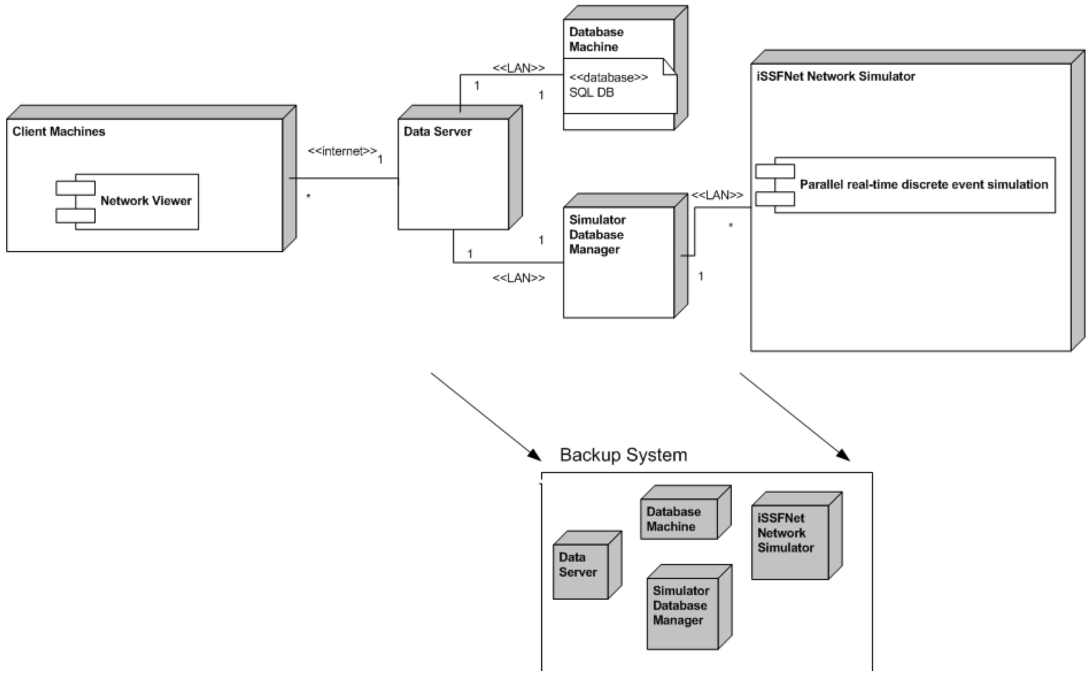


FIGURE 15 Deployment View Diagram

Within the iSSFNet Simulator Node (lower level):

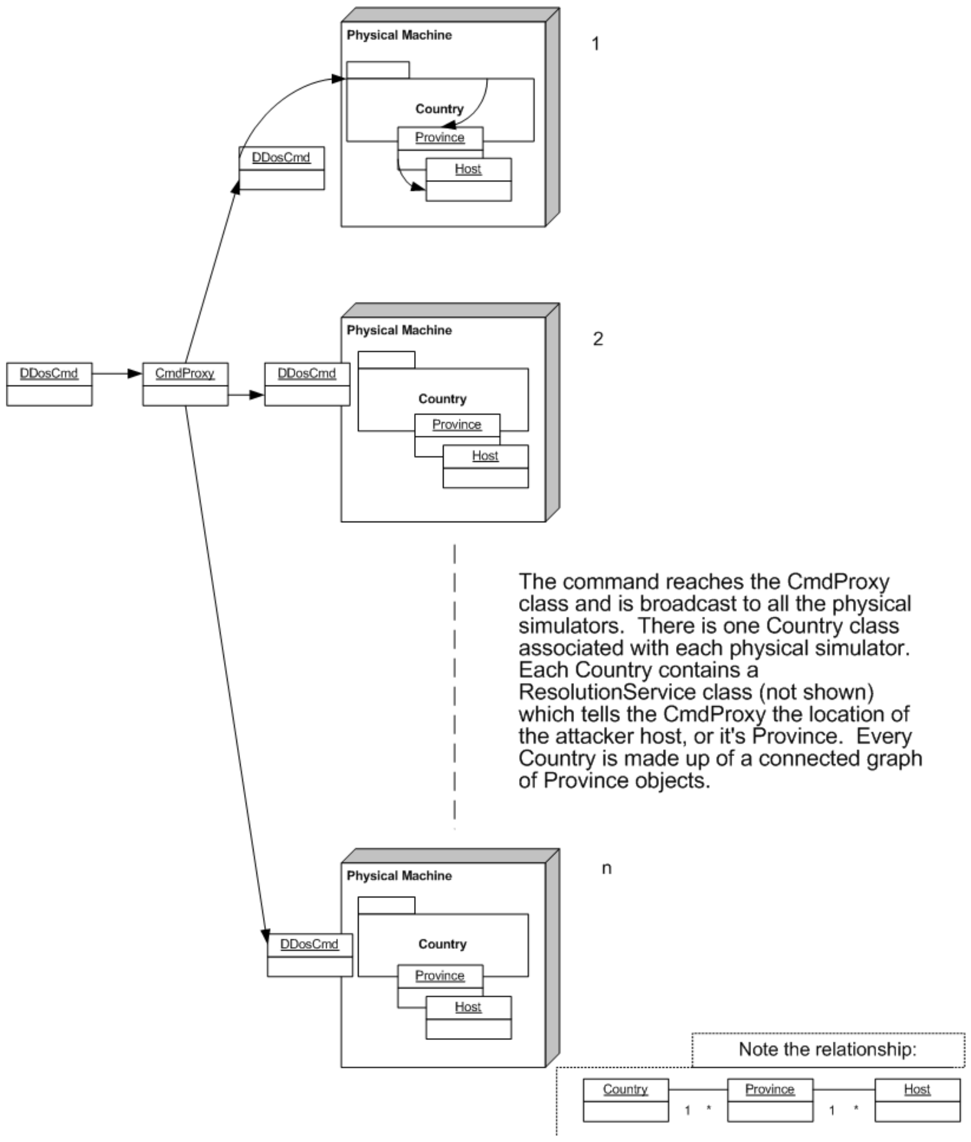

Physical Machine
1
Country
Province
Host
DDosCmd
Physical Machine
2
DDosCmd
CmdProxy
DDosCmd
Country
Province
Host
The command reaches the CmdProxy class and is broadcast to all the physical simulators. There is one Country class associated with each physical simulator. Each Country contains a ResolutionService class (not shown) which tells the CmdProxy the location of the attacker host, or it's Province. Every Country is made up of a connected graph of Province objects.
Physical Machine
n
DDosCmd
Country
Province
Host
Note the relationship:
Country
1 *
Province
1 *
Host

FIGURE 16 iSSFNet Simulator Node Diagram

Implementation View

This view provides information about the structure and contents of important source code folders.

ATTACK

All the classes for the simulation of the DDoS attack scenario reside in this folder:

The DDoSSession class is the core class that is responsible for initialization and configuration of the protocol session. It sets up the simulation timer and sets up protocol message and command objects. The DDoSMessage and DDoSCmd classes work with the DDoSSession class to communicate the commands for attack and report.

AUXILIARY

This folder contains the core classes of country, default auxiliary oracle, and resolution service:

The Country class instantiates the province, resolution service and command proxy objects. It is also responsible for performing the pre and post configuration tasks.

The DefaultAuxOracle class registers all the hosts and interfaces. It also sets up the in and out channels via which all the communication for a province occurs.

FLUID

This folder contains classes for handling fluid traffic at different network layers of IP and MAC. The FluidHdrMessage object specifies a basic implementation of fluid transport protocol message whereas the FluidAgentInPHY object defines a module at the physical layer to handle fluid related data. Similarly, the FluidAgentInMAC object handles the fluid traffic at the MAC layer. The FluidAgentInFilter object is responsible for processing an incoming packet. In case, it is a fluid dataflow event, its resource utilization would need to be taken care of as well to get around the fluid traffic loss. This would happen in case of CPU resource contention where the submission rate for packets to the upper layers would be constrained.

INTERACT

This folder contains various kinds of classes responsible for handling different kinds of commands, such as ping, ftp, shutdown, report, etc. The CmdProxy object makes sure that all the in and out channels have been established. All the tasks of reading and writing a command message get handled in this object. The CmdHandler object deals with the province and auxiliary oracle to process the event queue containing command messages.

NET FOLDER

This folder contains the core SSFNet classes for modeling and simulation of network elements of hosts, routers, network interfaces, and links. The configuration of arbitrarily complex network topologies is also performed by the classes residing in this folder. For example, one of the crucial task of this configuration involves the generation of traffic. In order to do this, the framework utilizes the statistical algorithms for internet traffic, such as the poisson and poisson pareto burst process algorithms.

OS

This folder contains the core SSFNet classes for modeling and simulation of network protocols, protocol messages, and operating system components. The Dijkstra class contains the implementation of the Dijkstra's famous shortest path algorithm. Support for internet class addresses has been provided via the Internet_Protocol class that implements the basic functionality of the IPv4 addressing scheme. The protocol of policy aware on demand routing has been utilized by the PAO_Routing class that helps the routers in deciding how to route the network traffic.

sOSPF

Provides classes which together implement a model of the Open Shortest Path First version 2 protocol (limited static version). The Open Shortest Path First (OSPF) protocol is an IP link-state routing protocol, recommended for distributing routing information among the routers in a single autonomous system (AS), with explicit support for classless inter-domain routing (CIDR) address allocation.

V.Quality Attributes

Performance

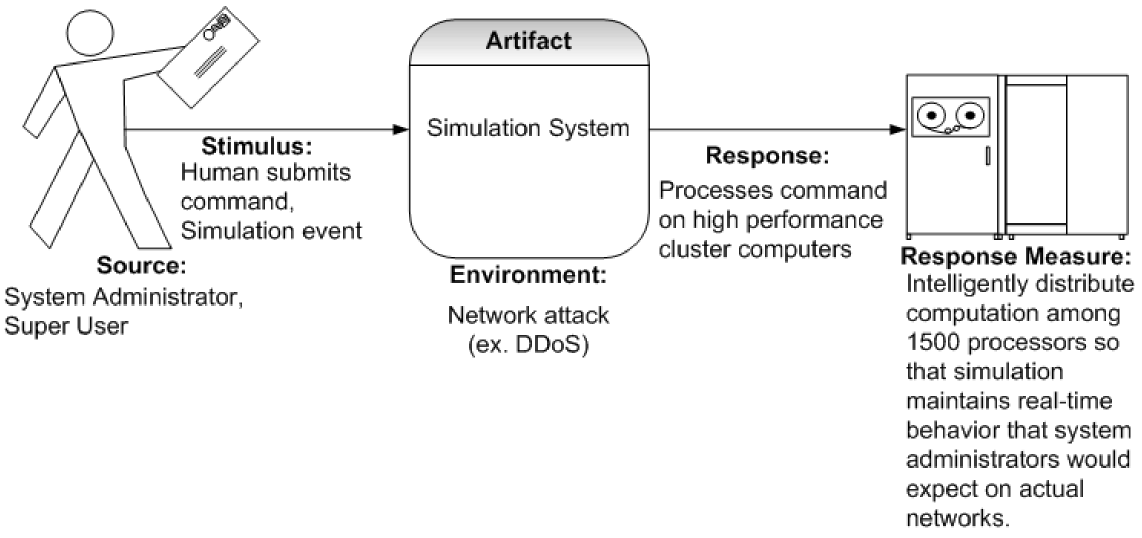

Artifact
Simulation System
Stimulus:
Human submits command, Simulation event
Source:
System Administrator, Super User
Environment:
Network attack (ex. DDoS)
Response:
Processes command on high performance cluster computers
Response Measure:
Intelligently distribute computation among 1500 processors so that simulation maintains real-time behavior that system administrators would expect on actual networks.

FIGURE 17 Performance Diagram

Performance is one of the most important quality attributes in the RINSE application. As stated earlier, previous SSF simulators existed with many of the same features as iSSFNet. However, these versions could not provide the performance necessary to simulate hundreds of large scale networks for many days. The RINSE team utilized a variety of architectural patterns and other novel tactics to achieve high performance.

Bass, et. all, suggest that "the goal of performance tactics is to generate a response to an event arriving at the system within some time constraint."[2, p. 111] This definition is very applicable to the iSSFNet system, a real time discrete event simulator. When events occur, they either execute normally or are delayed and latency is adversely effected. The intelligent use of hardware and software resources seeks to increase the amount of time in which events can be handled normally and minimize "blocked time." Resource availability hindered by failure or contention for resources can be fatal problems when many events hit a particular resource near simultaneously. The aforementioned problems can be dealt with looking at solutions in three categories: Resource Demand, Resource Arbitration, and Resource Management.

RESOURCE DEMAND

The RINSE architecture regulates event frequency as well as the resource consumption of individual events. This is accomplished through the use of algorithms and data structures which support computational efficiency. Additionally, an effort is made to reduce computational overhead through asynchronous parallel processing. Obviously, execution times and queue sizes are well bounded to prevent overruns.

Simulating the mechanics of network traffic routing is a non-trivial activity. Brute force implementations of routing information with n nodes requires $O(n^2)$ of memory. RINSE uses a novel hierarchical addressing scheme (BGP) which only stores IP prefixes. This policy based routing model permits on demand computation of routes. With the use of route aggregation, preloaded, pre-computed forwarding tables can compute routes for background traffic and other flows as required.

RESOURCE MANAGEMENT

Resource Management can be achieved by introducing concurrency, maintaining multiple copies of data, and increasing available resources. RINSE uses all of these techniques to manage resources. In the event that a hacker disables all or part of the RINSE network, a backup network at another location will receive the current state of the mainline system and continue the game. Increasing resources and concurrency are linked tactics. RINSE achieves them by employing high performance RISC processors with large distributed memories in the NCSA supercomputing cluster.

RINSE has also overcome some serious parallelization challenges. Normally, a global clock object would exist to synchronize the various machines working in parallel. Currently, RINSE runs on approximately 1500 processors. If a processor finishes its designated operation before other collaborating processors, it must wait. This is clearly inefficient and undesirable. RINSE has successfully implemented a scheme which involves local timers which run subsystems asynchronously. If this were done “brute force,” it would be much harder, perhaps impossible, to implement clever algorithms that simplify matrix mathematics for example. However, the RINSE team has successfully implemented their asynchronous system. The details of the implementation are beyond the scope of this documentation and likely not relevant to the reader.

RESOURCE ARBITRATION

Scheduling is the result of conflicts over system resources.The architecture of RINSE employs a variety of scheduling strategies to efficiently share resources.First, FIFO queues are used throughout the system, notably in the model for network interface cards.FIFO queues have the advantage of handling all resource request equally, scheduling events in order.Also, when there are multiple priority queues, such as within one single queue, a FIFO policy is used.

Fixed priority scheduling is also utilized in RINSE.For example, when the SSFNet kernel is handling an interaction or emulation task, current events are constrained by deadlines.In the situation handling incoming events, some events need to be processed and incorporated into the simulation immediately.Current work focuses on expanding the kernel to permit differentiation between lower priority tasks, such as background traffic computation, and higher priority tasks.Deadline monatomic strategies are also used in situations such as in emulation mode.When a “real” packet is converted to a “virtual” packet, a deadline time is imposed for delivery.Upon arrival at the simulated network card, the FIFO strategy can be violated to allow the packet to take its place in the queue such that its deadline is not compromised.

TRADITIONAL PATTERNS

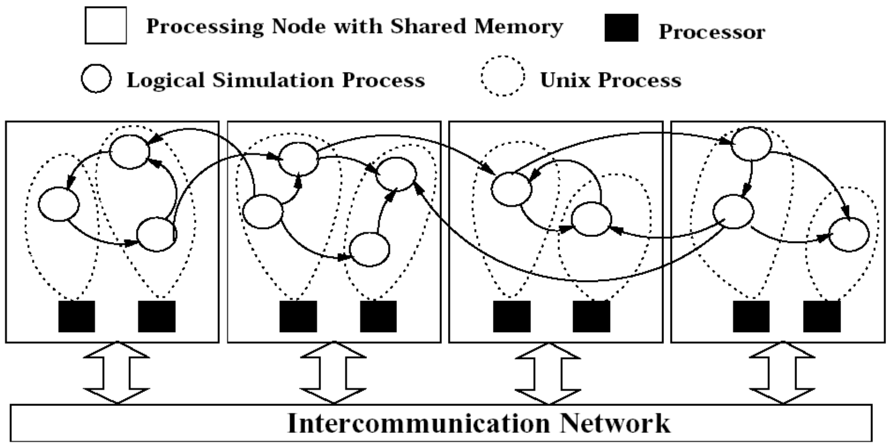
Layers Pattern

The Layers pattern describes the situation where logical decompositions each perform a subset of the total work, continuously passing responsibility down the chain until the operation is completed.[2, p. 205] This pattern provides many benefits, including keeping dependencies local to the individual layers, and this can be seen within the RINSE architecture.

RINSE employs a series of ProtocolSession layers, each representing a protocol layer in the ISO/OSI protocol stack. When a ProtocolMessage is created, each ProtocolSession performs whatever work it needs to, then calls ProtocolSession::push(msg) to push the message down to the ProtocolSession in the next layer. When the message reaches the bottom and all work is performed, each ProtocolSession can call ProtocolSession::pop(msg) to alert the ProtocolSession in the layer above that the message has been processed by the lower layer and should be bubbled back up.

Proxy Pattern

The intent of the proxy pattern is to “provide a surrogate or placeholder for another object to control access to it.”[7, pp. 207–217]


Processing Node with Shared Memory
Processor
Logical Simulation Process
Unix Process
Intercommunication Network

Sourece: The DaSSF User's Manual

FIGURE 18 Proxy Pattern Diagram

RINSE applies the solution found in the Proxy Pattern through the use of distributed memory.Large multiprocessor architectures cannot be implemented with the traditional single bus design.Bandwidth constraints limit the number of connected processors.In a distributed memory configuration, each processor has its own cache which has its own associated memory.However, the memories are connected through a network.

CUSTOM/FUTURE PATTERNS

By definition, a software design pattern is a time tested solution to a recurring design problem. As a result of the research which developed iSSFNet, a variety of new solutions to recurring problems in the network simulation domain have been developed. Strictly speaking, these are not patterns yet because they have not had time to gain industrial approval. However, these "patterns" are documented here in the spirit of sharing the knowledge.

Continuations Pattern

Summary: The Continuations Pattern is a new pattern developed to provide an efficient way to save state information and reduce latency in network simulation.

Context: You are developing a discrete event simulation.

Problem: How do you model a delay in the system and advance simulation time without losing state information?

Background:

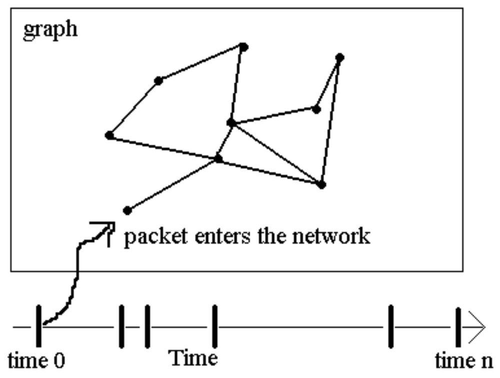


FIGURE 19 Continuations Pattern Diagram

Suppose you have a graph of your network topology. The network is composed of devices with well-defined interfaces. At time *x* in the simulation, a packet is injected into the network. At any point in time, it occupies a specific position in the graph. Now the simulation time must advanced to time $x + \Delta t$ to accommodate another event, perhaps an additional packet. However, if nothing is done, the current packet and the network state that was affected by it will be lost.

Solution: The solution to this problem is to define a continuation object. The purpose of this object is to store state information. Many times, the continuation object can just save a copy of the packet. One should also define a Timer object that contains a member variable which specifies how long of a delay should be modeled and encapsulates the continuation object.

Multiresolution Modeling Pattern

Summary: The Multiresolution Modeling Pattern is the principle technique that allows real time simulation of large possible. The fundamental concept is to be able to simulate network activity at different levels of detail depending on the particular needs of the particular scenario. This enables execution time to be reduced by two orders of magnitude.

Context: (a) You are developing a discrete event simulation, or (b) You want to model a variety of different situations, each requiring a different level of granularity.

Problem: How much should the process being simulated be discretized? Large granularity is useless for some scenarios, while fine granularity is unnecessary for others and computationally expensive.

Background: Suppose you are developing a simulator of computer networks. Depending on the type of attack or defense that a simulating network is exposed to, different resolutions of traffic modeling may be necessary. These vary from tracking at the individual packet level to observing aggregate flows between sub-networks.

Solution: The solution to this problem is to adjust the level of detail with which traffic is simulated:

> Traffic that is "in focus," what we call foreground traffic is simulated with high fidelity at packet level detail. Traffic that represents "other things" going on in the network, i.e., background traffic, is abstracted using fluid modeling, either fine grained per-flow models, or coarse time-scale periodic fixed point solutions.
>
> Fluid modeling of network traffic is a technique with some history, and is being explored also in other network simulators, such as MAYA, IP-TN, and HDCF-NS/pdns. The models used in iSSFNet are based on our previous work to develop discrete-event fluid modeling of TCP and hybrid traffic interaction models such that the packet and fluid representations can coexist in the same simulation.[3, p. 6]

Multiresolution traffic modeling provides for speedups of two orders of magnitude. This speedup facilitates the simulation of very large networks. The multiresolution configuration can be viewed this way:

Level 1: Model individual packets - Discrete
Level 2: Raise level of abstraction and look at traffic as a fluid - Continuous
Level 3: Raise the level of abstraction again and look at aggregate communication between sub-networks.

Each level reveals different information about the network under evaluation.

Flexibility/Extensibility

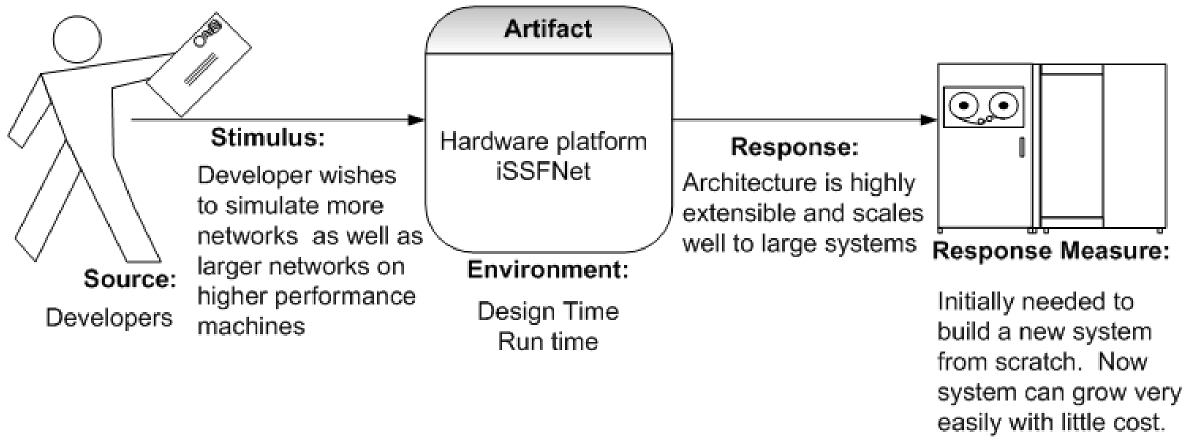

Artifact
Hardware platform iSSFNet
Stimulus:
Developer wishes to simulate more networks as well as larger networks on higher performance machines
Source:
Developers
Environment:
Design Time
Run time
Response:
Architecture is highly extensible and scales well to large systems
Response Measure:
Initially needed to build a new system from scratch. Now system can grow very easily with little cost.

FIGURE 20 Flexibility / Extensibility Diagram

RINSE was designed with flexibility, extensibility, and scalability in mind. The architecture needed to be intellectually accessible for new developers so that it could be improved in a straightforward manner. It also needed to be able to scale to run on large hardware platforms, such as over 1000 clustered high performance microprocessors.

In addition to the implementation of patterns, other techniques were employed to achieve this quality attribute. First, the APIs were redesigned from DaSSFNet to be more user friendlyfor new developers. Imposing additional structure does create more overhead, but the benefits outweigh the small performance hit. Other changes improved performance though, such as reducing the number of SSF channels. There is also additional support for emulation. Real time interaction is facilitated by enabling interaction with the outside world. Despite all these changes from the earlier SSF application, iSSFNet maintains backwards compatibility with old DML network models still work.

Bass, et. all, describe general tactics for modifiability. RINSE utilizes the "maintain semantic coherence,""anticipate expected changes," and "limit possible options" tactics to achieve modifiability.[2, p. 107] "Semantic coherence refers to the relationships among responsibilities in a module...The tactic of anticipating expected changes does not concern itself with coherence of a modules responsibilities but rather minimizes the effects of the changes."[2, pp. 106–107] In many ways, the SSF class of systems can be seen as a product line architecture. "Modifications, especially within a product line, may be far ranging and hence affect many modules. Restricting the possible options will reduce the effect of these modifications."[2, p. 107]

TRADITIONAL PATTERNS

Command Pattern

The GoF Command Pattern shows how to separate a request from its execution, such that you "encapsulate a request as an object, thereby allowing you to parameterize clients with different requests, queue or log requests, and support undoable operations."[7, p. 233] RINSE uses this approach to implement the commands which clients input with the Network Viewer client and send through the entire RINSE architecture. See the DDoS scenario in Section III to see how a command object flows through the system.

In SSFNet, a DDOSCommand object contains the parameters for a request (the DDOS attack source and destination addresses and the number of seconds between attacks), and the realization of these requests occurs as the simulator creates DDOSMessages using the parameters specified in the DDOSCommand object.

Strategy Pattern

The GoF strategy pattern's intent is to "define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it."[7, p. 315]

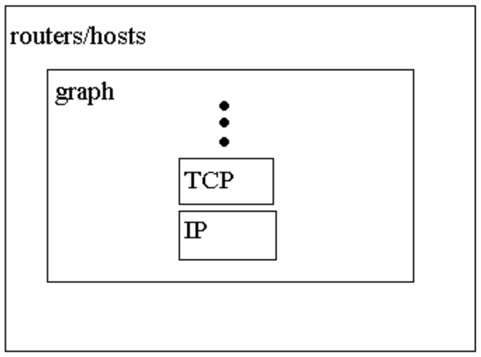


FIGURE 21 Strategy Pattern Diagram

In the example illustrated above as well as in many other instances, subclasses implement various strategies for a super class.

Templates Pattern

The intent of the Gang of Four (GoF) template pattern is to “define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.”[7, p. 325] This pattern is used in iSSFNet through the employment of the standard template library iterators. These iterators are used, for example, to iterate over nodes of topology objects.

WholePart Pattern

The Whole-Part pattern explains how to separate units into aggregate components, and the inherent aggregation of a network fits this perfectly.[8, pp. 225–242]In this case we have the Net class, which represents a network, containing Hosts, which are computers on the network, containing Interfaces, which represent the network interfaces within a computer. Also note that the Net class can consist of child Net classes (subnets), each with their own Hosts with their own Interfaces.

Custom/Future Patterns

Fractal Design Pattern

Note: This pattern gets its name from fractals found in mathematics because they are defined by self-similarity.

Summary: By structuring the simulation objects in a way similar to the real world objects under investigation, domain experts can easily learn how the simulation works and extend it if they wish.

Context: (a) You are developing a discrete event simulation, or (b) Other software engineers who understand the class of systems that is being simulated but not simulation techniques must modify or interact with your simulation.

Problem: How do you efficiently organize the simulation so that the engineers mentioned above can quickly learn the system?

Background: Suppose you are developing a simulator of computer networks. You want computer network experts to interact with your system and develop models to enhance it. The network experts know nothing about discrete event simulation.

Solution: The solution to this problem is to structure your simulation software similar to the structure of the system you are modeling. The classes in your simulation should be named after the objects they are simulating. Therefore, if you understand the internet and computer networks, you can understand the network simulation software. This concept is an extension of the paradigm behind object oriented program which states that classes should be named after the real world objects they control.

Subclass Creation Pattern

Summary: The RINSE designers have implemented an unusual process by which subclasses such as Command, Protocol, and Session call associated register() functions, i.e., Command subclasses call Cmds::registerCommand(), Protocol subclasses call Protocols::registerProtocol(), Message subclasses call Messages::registerMessage(). The arguments to these functions are a Constructor function pointer, a string identifying the Class name, and an integer identifying the Class. These methods inject the constructor function pointer and the class name into a map; then when the app needs to construct one of the subclasses, it calls an associated newInstance() function to initialize the new class, i.e., Cmds::newInstance(string className) or Protocols::newInstance(string className).

Context: You have multiple subclasses that the system needs to be aware of.

Problem: How do you keep track of the diverse subclasses when you need to know which actual instance-type to instantiate?

Solution: Keep a map of all available subclasses. At runtime, have each subclass add itself to the map so that the application can be aware of it and use it as needed.

6. Conclusion

RINSE is an outstanding example of a well architected software system. Through the use of architectural patterns and other quality attribute focused tactics, the development team made significant progress in furthering the areas of:

1) Multiresolution traffic modeling
2) Real-time interaction
3) Efficient routing simulation
4) High performance parallel simulation

The success of RINSE and specifically the iSSFNet component can easily be attributed to the domain knowledge of the architects as well as expertise in object oriented design. This project is an excellent case study in software architecture done properly in a research organization.

In this paper we have attempted to demonstrate a rigorous software architecture documentation approach based on the principles explicated by Bass, Clements, and Kazman. Our approach began with the architectural business cycle. We documented important information on the stakeholders, the development organization, the technical environment, and the architects' experience. Later, we examined the salient architecture information such as important classes and in this case, the distributed denial of service attack scenario. Next, we presented a number of architectural views, e.g., logical, module, component/connector, deployment, and implementation. To conclude, we explored key quality attributes such as performance (e.g., resource demand, resource management, resource arbitration, traditional patterns, and custom patterns), and flexibility/extensibility.

Acknowledgments

The authors are grateful for the guidance and wisdom of Ralph Johnson as well as the willing participation of the RINSE development team.We also wish to acknowledge the significant contributions of Adnan Zaman to this research.

Conflict of Interest

The authors declare no potential conflict of interests.

References

[1] E. Woods, “Software Architecture in a Changing World,” *IEEE Software*, vol. 33, no. 6, pp. 94–97, Nov. 2016.

[2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. New York: Addison-Wesley Professional, 2003.

[3] M. Liljenstam, J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier, “RINSE: The Real-Time Immersive Network Simulation Environment for Network Security Exercises,” in *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, Washington, DC, USA, 2005, pp. 119–128.

[4] P. Gustavson, K. L. Morse, R. Lutz, and S. Reichenthal, “Applying Design Patterns for Enabling Simulation Interoperability,” presented at the Spring Simulation Interoperability Workshop, 2004.

[5] H. Neu and I. Russ, “Reuse in Software Process Simulation Modeling,” presented at the Software Process Simulation Modeling Workshop (ProSim 2003), Portland, OR, 2003.

[6] U. Klein, T. Schulze, and S. Strassburger, “Traffic simulation based on the High Level Architecture,” in *1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274)*, 1998, vol. 2, pp. 1095–1103.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley Professional, 1994.

[8] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, vol. 1, 5 vols. New York: Wiley, 1996.