WEB SERVER PERFORMANCE SIMULATION: FINAL REPORT

ANDREW FERRIER
Supervisor: Peter Harrison

Department of Computing
Imperial College
London

Version 1.0
(software version 1.0)

Abstract

This report documents my 4th Year MEng Individual Project, which was on the subject of 'Web Server Performance Simulation'. There were two parts to the project:

The first part involved writing a program, which I called WS^3 (Web Server Simulation System), using the Java programming language, which allowed the simulation of a generalised web serving system by accepting a specification for it (specifying clients, the network, and servers) in XML (eXtensible Markup Language). The initial design for WS^3 was exceeded and I added many extra features. WS^3 is available for free use [1].

The second part involved using WS^3 to investigate some general issues of web server performance and setup. I came to the conclusions that: the processor/thread ratio on a server greatly affects the client response time but that this relationship is complex and simulation software such as WS^3 should be used to investigate specific situations; the question of whether 'large' or 'small' servers are more efficient is difficult to answer and requires detailed simulation; the use of different statistical distributions in a program such as WS^3 can produce unexpected results and they must be chosen with care; the speed of simulations such as WS^3 depends greatly on the number of objects being simulated and that this speed is directly proportional to the number of those objects; and that use of the Pareto distribution can improve the realism of a simulation but means that it can take a long time to converge to equilibrium.

Acknowledgements

I'd like to acknowledge the help of my supervisor, Peter Harrison [2], in the execution of my project. His comments on my project were encouraging and helped me enhance an idea which I was already keen on.

I'd like to thank Uli Harder [3], who was a supervisor for the first half of my project, who gave me much valuable input and useful ideas, and explained areas I should give attention to.

I'd also like to thank Tony Field [4], for allowing me to use and modify his simulation library, and for helping me to arrange my project supervision originally.

Finally, I'd like to thank my personal tutor — Fariba Sadri [5] — and my friends and family, for helping me through the last four years at Imperial College.

Contents

I The Project **11**

1 Introduction **13**
- 1.1 Aims . . . 13
- 1.2 Project Code and Reports . . . 14
- 1.3 Knowledge Assumed . . . 14
- 1.4 Report Outline . . . 14

2 Background **17**
- 2.1 Performance for Clients and Servers . . . 17
- 2.2 Current Practice and Research . . . 18
- 2.3 Why Use Simulation? . . . 20

II The Software **23**

3 Specification **25**
- 3.1 Summary Specification . . . 25
- 3.2 Input . . . 25
- 3.3 Simulation and Execution . . . 27
- 3.4 Output . . . 28
  - 3.4.1 Summarised Output . . . 28
  - 3.4.2 Trace Output . . . 28

4 Design **29**
- 4.1 General Design Assumptions . . . 29
  - 4.1.1 Internal Design Assumptions . . . 29
  - 4.1.2 External and Interface Design Assumptions . . . 30
- 4.2 Class Structure . . . 31

5 Implementation Issues **35**
- 5.1 Programming Language for Implementation . . . 35
  - 5.1.1 Input File Format . . . 36
- 5.2 Changes to Specification and Design . . . 37
- 5.3 Java Simulation Toolkit . . . 38
- 5.4 Randomness . . . 39
- 5.5 Equilibrium . . . 41
- 5.6 Documentation . . . 42
- 5.7 Standards . . . 42

6 Testing **43**
6.1 TST-001 . . . 43
6.2 TST-002 . . . 46
6.3 TST-003 . . . 46
6.4 TST-004 . . . 47
6.5 TST-005 . . . 47
6.6 TST-006 . . . 48
6.7 TST-007 and TST-008 . . . 48
6.8 Non-File-Suite Testing . . . 49

III Uses for the Software **51**

7 Evaluation **53**
7.1 Notes About the Input Files . . . 53
7.2 Web Serving Guidelines . . . 53
7.2.1 Size of Machines Used . . . 53
7.2.2 Server Threading . . . 56
7.2.3 Distributional Differences . . . 57
7.2.4 Real-world Example . . . 59
7.3 WS^3's Speed . . . 61
7.4 WS^3's Accuracy . . . 63

8 Conclusions **65**
8.1 Construction of WS^3 . . . 65
8.2 General Conclusions about Web Systems and WS^3 . . . 66
8.3 Strengths and Weaknesses of WS^3 and the Project . . . 67
8.3.1 Strengths . . . 67
8.3.2 Weaknesses . . . 67
8.4 Future Extensions . . . 68
8.4.1 WS^3 Features . . . 68
8.4.2 Analysis of Web Serving . . . 69

IV Appendices **71**

A User Guide **73**
A.1 Licencing . . . 73
A.2 An Overview of WS^3 . . . 73
A.3 System Requirements . . . 74
A.4 Installing WS^3 . . . 75
A.4.1 Installing on Unix/Linux . . . 75
A.4.2 Installing on Windows . . . 75
A.4.3 Warning About Installation on any Platform . . . 76
A.5 File Structure Once Installed . . . 76
A.6 Testing the Install of WS^3 . . . 76
A.6.1 Testing on Unix/Linux . . . 76
A.6.2 Testing on Windows . . . 76
A.6.3 Randomisation Test . . . 77
A.7 Rebuilding the Javadoc Documentation . . . 77

A.8 Creating an Input File 77
A.8.1 Points to be Noted about Input Files 77
A.8.2 Input File Elements 78
A.8.3 System Objects 79
A.8.4 Connections and Routes 81
A.8.5 Distribution Types 83
A.8.6 Time-to-live Expiry 85
A.8.7 Constraint Enforcement 85
A.9 Running WS^3 85
A.9.1 Data Dumping 86
A.9.2 Tracing 86
A.10 Interpreting the Output of WS^3 87

B Network Diagram Conventions **89**

C UML Diagrams **91**
C.1 Conventions 91

D Bibliography **93**

E Glossary **99**

F List of Acronyms **101**

Part I

The Project

Chapter 1

Introduction

1.1 Aims

This project was about investigating web server performance. I did this by constructing software, which I call WS^3 (Web Server Simulation System), to simulate web serving systems, and then using this software to simulate fictional systems and analyse them.

Web Server Performance is an important issue. More and more organisations are setting up web servers, and many are notorious for producing inadequate performance [6, 7]. Sometimes this lack of performance is due to overly large or complex pages, sometimes it is due to poor network (Internet or intranet) connectivity, but sometimes it is also due to inadequate server configuration, whether that be the hardware or software[1].

The latter issue interested me, and I wanted to see if I could come up with a tool and also some general guidelines for configuring web serving systems. Thus I decided to do this project, and wrote simulation software to help in making server configuration decisions.

I had two primary aims for my project:

- To write a program to enable a user to simulate an a web server system, which consists of web servers, clients, and a network. This is documented in part II.

- To use this software with some fictional web-server system specifications to come up with some general guidelines and answers to questions about the most efficient way to setup and configure a web serving system. This is documented in part III.

  There are many questions which I would have liked to answer about web serving systems. I decided to pick a few specific ones which I would attempt to answer, investigating other aspects if time allowed. The primary questions I decided to investigate were:

  - Whether it is more efficient and reliable, generally, to use many small interconnected serving machines (a web server 'farm' [8]), or to use one or only a few large serving machines.

[1]Background issues are discussed in greater depth in chapter 2.

  – How many synchronous threads should exist on a single server system for maximum performance.

These questions are studied in chapter 7 (Evaluation).

1.2 Project Code and Reports

This project has a website, with all of my project reports (the outsourcing report, the progress report, and this report) available for download (see [1]). Source code for WS^3 is not supplied in this report, but both source code and compiled executable code are available for download from the website in a single package. Contact details are also available there.

For Department of Computing users, all of this content is also available in the directory `~ajf98/project/`.

The latest version of the software at the time of writing is 1.0.

1.3 Knowledge Assumed

This report assumes the reader is familiar with:

- What a web server and client are; the differentiation between them and how they work together to form the WWW (World Wide Web), as well as what the TCP/IP (Transmission Control Protocol/Internet Protocol) and HTTP (Hypertext Transfer Protocol) protocols are and how they work generally [9, 10]. However, detailed knowledge about the TCP/IP and HTTP protocols is not assumed.
- The basic principles of simulation and modelling; the difference between process-based and event-based simulation. Also, the basic principles of queueing theory [11, 12].
- An understanding of object-oriented programming and Java [13].
- The basic syntax and principles of XML [14]. This is necessary to understand the specification for the input file to the software. XML is very quick and easy to understand however.
- The basic syntax of UML (Unified Modelling Language). UML diagrams are used in this report. A basic introduction to UML, with links to more information, can be found in appendix C.

1.4 Report Outline

In this report, there are the following chapters:

Chapter 2: Explains the background to my project: the problems and issues which led to my decision to do this project, the current state of progress in this subject area, and why I used simulation.

Chapter 3: Provides a high-level specification for WS^3; similar to the specification found in the outsourcing report [1].

Chapter 4: Explains the design for the internal software structure of WS^3.

Chapter 5: Explains the issues that arose during the implementation of WS^3.

Chapter 6: Describes the testing that I applied to WS^3.

Chapter 7: Investigates the questions posed in section 1.1 on page 13, and discusses the speed and accuracy of WS^3.

Chapter 8: Summarises what the project achieved; the construction of WS^3; and the conclusions reached with the aid of WS^3.

The appendices contain, amongst other items, a glossary and the bibliography.

Chapter 2

Background

In this chapter I investigate the background issues behind this project[1], by:

- Discussing performance issues for clients and servers of the WWW.
- Looking at the current state of progress in the area of web server simulation, and related areas.
- Explaining why I am using simulation as a tool to explore web server performance.

2.1 Performance for Clients and Servers

Web Server Performance is an issue that has come to prominence in the last decade or so, since the invention of the web in late 1990 by Tim Berners-Lee at CERN [15].

There are two main ways that one can look at web server performance [16]:

- From the user (client)'s perspective.
- From the server's perspective.

From a user's perspective, all that matters about web server performance is the speed with which web pages are retrieved and how they then display on their client.

In fact, it has been shown that for client/server systems in general (not just the web), sub-second response items to user actions are near-essential for users to feel that the system is responding quickly enough not to interrupt their flow of thought [17]. Even if one doesn't agree with this, it is obvious that users prefer short response times and it will benefit all concerned if these occur [18].

From a server's perspective, what matters is the ability of the server to handle as many requests as clients demand, synchronously, without unnecessarily delaying responses to the clients (hence causing them to become frustrated), or building up a backlog of clients wishing to use the server.

[1]Some of the material from this chapter was taken from my outsourcing report, available at the project website [1].

Obviously ensuring fast response times for clients and sufficient server capacity are not totally distinct problems: to have a coherent 'World Wide Web' which works for everyone, performance has to be good all round.

Hence, in this report, and this project, I am interested in two related issues:

- The total response time for a client when making a request.
- Sizing and capacity planning for the web server, to ensure that the client response time is low, and efficient use is made of the server resources.

Both of these depend on two general factors: the network in-between the client and the server, and the capabilities of the server hardware and software[2].

In this project I investigated both of these factors, but the stronger focus was definitely on the server capabilities. Network performance is a very large topic in itself, and it is worth bearing in mind that even TCP/IP networks carry many different types of traffic, not just HTTP, and hence network performance issues are not just issues with HTTP. This is a subject worthy of investigation all by itself and so one I didn't attempt to embark upon in my project.

Note: When I refer to client-end performance, I will *not* be looking at performance after pages have been retrieved. The ability of web browsers to perform animation, run client-side code, or use interactivity front-ends such as Macromedia Flash [19] is not within the remit of this project.

2.2 Current Practice and Research

Computer-based simulation is a fairly well-studied subject, and there are many books, papers, and other sources, which document the area from many perspectives. One of the better ones is [20], a fairly modern and practical book which I have found quite useful. However, there are many others, of which [21, 22, 23, 24] are only a few. There are also statistics books specifically geared towards creating computer simulations — for example, [25] has some fairly technical and thorough information on various different kinds of distributions.

Performance simulation, in particular, has been an area of research of some time, because simulation is often an ideal tool for assessing and evaluating performance [26, 27].

It does not appear that much progress has been made in the area specifically of web server simulation, however[3] — one of the reasons why I decided to do this project, and the main reason this section is sadly lacking in volume! There are simulation tools which could be used to model web serving systems, such as some of those listed by the Simulation Software Survey [28], but most of these suffer from these flaws:

- They are too general, not being specific to web serving. I see this as a flaw because it means that more work has to be done to create a simulation which models a web serving system.

[2]Normally the capabilities of the client are irrelevant because even the simplest client will have the ability to receive traffic as fast as it can be sent by the network and the server, and clients typically only retrieve a few files simultaneously. Thus I did not model the client capabilities with WS[3] extensively.

[3]Ironically and 'unhelpfully' (at least when searching for information!), there seem to have been many people who have tried *web-based* simulation — but not related to actual simulation of web systems themselves.

- They have a graphical front-end; this is something I wanted to avoid in creating the simulation software. An example of a tool with a graphical front-end is Tony Field's SwingSim application [4] — which allows one to create simulations using his simulation toolkit (which I used for WS^3), with a GUI (Graphical User Interface) Java drag-and-drop interface. This kind of tool has it's uses, but it was not what I wanted to create for my project, because:
  - SwingSim is primarily designed for understanding basic queueing systems; the user can very easily change parameters and re-run the system. I wanted to create a tool with a very specific purpose: as a web server sizing tool.
  - SwingSim has a drag-and-drop GUI interface. I wanted to create a tool with a batch interface: create the input file, run it through the simulation software, read and analyze the output. The drag-and-drop interface is easier to use, and is ideal for learning about queueing systems and queueing theory, but will not scale to very large or complex systems, whereas I wanted to create a tool that could do this — where one could easily specify hundreds or thousands of clients in the system, for example.

Another collection of the simulation software is the listing of *Simulation Software Links* [29] — most of the software listed are fairly low-level toolkits designed for constructing simulation software, rather than simulation software in itself. Also, most simulation software uses event-based simulation, whereas I wanted to use process-based simulation because I find it considerably more intuitive and hence, for myself, I believe that faster development results.

Web server performance appears to be a subject which is discussed in some circles [30, 31], but this does not often stretch to simulation. One example of a web serving simulation that has been attempted is [32]. However, this source does not contain much information on the simulation itself or how it was implemented — certainly not enough to compare it to what I have done — and the simulation is not freely available.

Even research in Journals such as the *ACM Transaction on Modeling and Computer Simulation* [33] does not turn up much information on web server simulation — it simply seems to be a subject of little interest, possibly because it is seen as too complex a way of server sizing for practical purposes. For many short-lived projects, this may well be true, but my industrial experience teaches me that projects often exist for longer than originally planned. Thus I think it is important that creation of a server sizing (also known as capacity planning) tool for web server systems be attempted.

Progress is being made in related areas. For example, queuing theory has been used as a tool to model web servers [34], but as I will explain below, this method can have disadvantages. Papers have also been published on profiling web usage analysis [35], which is a subject area which can have an impact on web server sizing, in particular for large projects. This book is quite useful if one is interested in understanding what users are using a web server and what they are using on it, which of course one should be if setting up a web server, but does not help with the direct problem of how to size the web server — in a way, knowing about the profiles of your users helps with two other problems: who to set up your network links to, and what to put on the site itself.

Work has been published on engineering software for the web (i.e. how to structure programmatic web-sites) [30]. This is once again useful to those attempting to size web servers as it helps them to understand what programmatic demands sites will place on their servers, but it is not of direct relevance to web server modelling.

Progress is also being made in the area of data mining and retrieval on the web — [36] is but one example. These are traditional subject areas which have been around for a while and are now being investigated in the context of the web. [37] contains information on servers farming out search tasks to child servers, for example. However, this has more to do with search engines, web indices, and other related technologies, then the sizing and creation of web server(s).

So, although sadly there is not much to document in the area of web server simulation itself, this was the main reason that I felt that it would be useful to try to write a web server simulation myself. Whilst I did not hope to make it in any way definitive or complete, I did document the improvements that would have to be made to further improve it (see section 8.4 on page 68).

2.3 Why Use Simulation?

In order to explain why I used simulation, as opposed to other methods of analysis, such as direct mathematical modelling using queueing theory, to investigate the performance of web servers, I will examine four potential strategies which can be used when deciding how to size web servers (of which only the first two tend to be used in practice):

1. Use rules-of-thumb; calculate using approximate formulae but always overestimate to make sure one has enough capacity. This is the approach normally taken in industry, and is easily subject to obfuscation and confusion, since it is based on empirical measurements. This observation is based mainly upon personal experience.

2. Use real measurement; set up a real system, with a real web server and some real clients, attached with a network, and install software on them to make many simultaneous requests for files. This is a typical method for benchmarking web serving systems. Examples of software that do this are Apache's Flood tool [38] and `httperf` [39]. These do, to a certain extent, help with deciding the load that a server can handle, and are sometimes used for larger projects. However they have several disadvantages:

    - Instead of having many thousands (say) of distinct clients, one will typically have tens or maybe hundreds, making many more requests than they typically would to produce the same demand. However, one has to be absolutely certain that these clients can handle the load of making that many requests. A typical client only handles a few requests simultaneously, which is trivial. If the client handles more than this, it may become bogged down in network code and will hence not produce realistic timings — in other words, one cannot necessarily extrapolate the results as one would desire.

   - Typically such machines are connected together on a LAN (Local Area Network) because it is prohibitively expensive to set up a more realistic network. This is unrepresentative of most web serving systems: although it may reliably test the load on a server designed only for a small intranet, any server which is serving requests to clients which are widely geographically spread will have much higher latencies on requests and replies because packets will have to travel through many switches and routers, possibly encountering delays along the way, outbound and inbound. This will hence increase transaction time for web requests, meaning that server resources are used up for a lot longer than they are when the clients are on a local LAN and making the test misleading.
   - Sometimes it is not possible to alter a running system to experiment with different configurations as it is too important — obtaining duplicate hardware or a creating a duplicate environment may not be an option.

   Some of these problems can be overcome: for example, it can be arranged for geographically spread clients to test the server. However, whatever one does, all real-world testing is very expensive and time-consuming, and becomes more so the more realistic one attempts to make it, in general [40]. Moreover, for the reasons explained above, it may well produce misleading and hence possibly dangerous (if the server sizing procedure is important) results.

3. Use queueing theory to investigate the system. This is very good in theory; however the problem is that it is very mathematically demanding, requiring quite a lot of mathematical knowledge, which is typically not available to those setting up web serving systems, and more importantly, most non-trivial systems cannot be solved exactly or easily. Sadly, this means that it only has limited application for most real-world systems and it is not generally used for entire systems — only for sub-systems, and then often in an informal way [41].

4. Use simulation software to model the system. This is the approach I will take with this project, and I will explore this further below.

Simulation software has several advantages over the set-it-up-and-see approach, and I will outline those here in the context of my project:

- One does not require the expense and inconvenience of setting up an actual system: this can take a long time [11, 40] and be very resource-intensive. In many cases, it simply would not be done — simulation thus allows one to explore new possibilities.

- Simulation is a lot less risky than altering an already-existing system: simply set up a 'duplicate' system in the simulation, then alter some of its attributes.

- It can answer many different questions, easily and quickly, e.g. What would happen if the attributes of this server were changed? What about if I double the number of servers? What is the mean value for this wait time? And so on . . .

However, simulation software also has some disadvantages:

- It does not produce exact solutions to most problems; solutions are at best approximate. More detailed simulation models typically produce more accurate answers, but it is still difficult to determine the absolute accuracy of those answers. I tried to overcome this by making the simulation model as specific and detailed as possible. My project being fairly short, I could not make it as detailed as I would have liked, but I tried to be aware of this shortcoming. I also discuss accuracy in section 7.4 on page 63.

- Dependent on the simulation software and hardware, determining solutions of sufficient accuracy can be very time-consuming and resource intensive, more so than running the corresponding real system. I was not to be overly concerned about this, and tried to overcome this problem with good quality software engineering, hence making my program as fast as possible. However, whenever it came to a tradeoff between the simulation quality and the speed of the software, I chose the former — I decided I simply did not have enough time to focus on the performance of the software itself as well as the other project objectives (though I did do a small amount of investigation into the performance of the software — see section 7.3 on page 61).

- Simulation software only tends to answer specific questions. Finding general answers and relationships can be difficult. This is something I was very aware of in my project when I tried to evaluate WS^3 — see chapter 7 for more information on this. Nevertheless, simulation software is very useful for specific practical situations, which is one of the reasons I wrote WS^3.

Part II

The Software

Chapter 3

Specification

This chapter gives a specification for WS^3 in terms of the input, the output and the expected behaviour. It doesn't contain information on the design of WS^3 — that it is in chapter 4. This specification was originally written for my outsourcing report, and I have duplicated it here, but modified it appropriately for WS^3 as it now stands[1]. It is written as a specification, assuming that WS^3 does not yet exist, and thus does not refer to WS^3 by name.

Not all features and details of WS^3 are described in the specification in order to keep it to a reasonable length, but the important ones are.

3.1 Summary Specification

The simulation software should simulate an arbitrary client-server system based upon the WWW model, i.e. with clients requesting files from servers via the HTTP protocol.

The simulation program must be written so as to accept input as specified in section 3.2, execute as per the description in section 3.3, and produce output as per section 3.4. This specification avoids discussing issues such as the language to use for the software and the look and feel of the software (for example, whether it is a console-based application or a GUI application), and focuses on the behaviour.

The basic simulation system as specified here simulates three types of 'system objects' — clients, network nodes, and servers. They interact with each other in the simulation and various parameters from them are measured and should be output by the simulation software.

3.2 Input

The input to the software should consist of a single XML [14] file which should specify the complete system. The name of this file shall be specified by the user to the software. The XML file must be valid according to the XML 1.0 specification [14] and must conform to the WS^3 XML Schema supplied with the

[1]Information on how the specification and design was changed is available in section 5.2 on page 37.

software[2]. Because of this requirement, only an outline of the specification is given here for the input file because the XML Schema specifies it completely.

The XML file will have the following structure:

- `<system>`. This element represents a simulation system. There must be only one of these elements in the file — all other elements are contained within it. The element is provided only for future-proofing, in case one wanted to develop the software to allow more than one simulation system to be specified in the same XML file. This feature is not provided in the software.

There are then various elements which can appear enclosed within the `<system>` element, which specify objects in the simulation system (these objects are known henceforth as system objects). These elements are:

- `<client>`. This element represents one or more instances of a client system. The element must have one and only one `<creationDistribution>` sub-element, which represents the inter-event time distribution with which the client makes HTTP requests (in virtual seconds). Requests can be synchronous: a client can make one request and then make another whilst the first is outstanding.

- `<server>`. This element represents one or more instances of a server system. The element must have one and only one `<serviceTimeDistribution>` sub-element, which represents the service time for an incoming HTTP request (in virtual seconds). The server object has the following attributes unique to it:

  - `threads`, `processors`, `threadGrain` — For more information on these attributes, see sub-section A.8.3 on page 79 of the user guide.

- `<networknode>`. This element represents one or more instances of a network node. This element must have one and only one `<serviceTimeDistribution>` sub-element, which represents the distribution of a packet's wait time in the node before being sent on (in virtual seconds). Packets are dealt with in a FIFO (First In First Out) manner.

All of the above system objects have the following common attributes and elements:

- `<name>`. An element which identifies the system object. This must be a unique identifier within the system (not just within this type of object) to enable inter-system-object connections to be specified (see the common element `<connectto>` below). If it is not unique, the software can report it as an error. This parameter is a non-zero-length string. This attribute is compulsory.

- `instances`. This attribute specifies the number of instances of this system object within the system. This attribute is an integer which is greater than 0. The default if this attribute is not specified is 1.

[2]This is called `ws3.xsd`.

The elements `<creationDistribution>` or `<serviceTimeDistribution>` are used within the `<client>`, `<server>`, and `<networknode>` elements to specify statistical distributions, and must have one (and only one) of the following as a sub-element:

- `<constant>`
- `<exponential>`
- `<uniform>`
- `<geometric>`
- `<erlang>`
- `<positiveNormal>`
- `<weibull>`
- `<pareto>`

For more information on the syntax of these elements, see the XML Schema supplied with WS^3. For more information on how to use them and what they mean, see sub-section A.8.5 on page 83 (in the user guide).

The element `<connectto>` can appear zero or more times inside the elements `<client>`, `<server>`, and `<networknode>`, and contains the unique system name referring to the object that one wants to connect to. This creates a one-way network connection (if two-way is desired). The element `<routeto>` creates a route between two system objects. For more information on the use of these elements, see the user guide (sub-section A.8.4 on page 81).

3.3 Simulation and Execution

The simulation is initially set up as described in the XML file, so that all system objects behave at all times as specified there.

The clients generate HTTP requests according to the distributions specified for them, and send them out onto the network. The network nodes queue incoming packets and send them onwards, delaying them by their delay distributions. Servers queue incoming requests, serve them according to their service time distributions, and send replies back to the same client which made the request — the replies are dealt with by the network nodes on the same basis as requests. Clients do not queue incoming replies.

Servers with more than one thread execute the threads in parallel, in virtual time. There is only one input queue for the server, and the threads retrieve the next request from the head of the queue when they are not busy.

All time is measured in virtual seconds, starting at 0 when the simulation is started, and the system is executed 'as fast as possible' (which may be slower or faster than real-time depending on the performance of the implementation).

Actual requests and replies do not need to consist of anything.

Routing is done according to the connection and routing information provided by the user in the input specification. If two routes are possible, the one the software chooses is undefined.

If there is no possible route from the source to the destination, the software outputs a warning and 'drop' the packet.

3.4 Output

3.4.1 Summarised Output

The following items must be output from the software:

- For each client: the number of requests sent; the number and percentage of replies received; the average response time and the variance of the response time.
- For each network node and server: the mean utilisation (load) on the system object. Also, the mean service time for the system object, and the mean time each message spent in the queue. The length of the queue on the system object at the end of the simulation; the number of requests processed and replies sent.

3.4.2 Trace Output

As well as the above summarised output (which should always be produced), the software should produce trace output. Whether this trace output is produced or not should be a user-configurable option.

The trace output should contain information on every discrete event that happens in the simulation system. Exactly what information is outputted is left as an implementation issue, but at least the following items should be included for each event:

- The unique name(s) of the system objects involved.
- A descriptive name describing the event.
- The virtual time at which that event occurred.

The trace output should be sorted strictly according to increasing virtual time.

Chapter 4

Design

This chapter describes the design of WS^3 — the 'pure' software design that I did before I started to write program code for WS^3. Not all aspects of the design are covered to ensure conciseness but enough is described to give a high-level view[1].

Note: Several diagrams in this chapter show abstract class hierarchies, and other design features of the WS^3 software. These diagrams are shown in the UML graphical modelling language. A very quick introduction to UML is provided in appendix C on page 91. Also, UML conventions as used in this report are explained in section C.1 on page 91.

4.1 General Design Assumptions

These were the design assumptions I initially made when I was designing the system. They are presented primarily to simplify the design of the system and attempt to keep it consistent. These are in addition to the behavioural specification outlined for the simulation in section 3.3 on page 27. They are outlined in a forward looking sense; as if WS^3 does not yet exist. There is necessarily a small amount of repetition in this section from the specification (chapter 3).

4.1.1 Internal Design Assumptions

- The system is essentially a queueing network, with varying delays at various points in the system, determined by statistical distributions which I will put into place.
- The system is constructed of one or more clients, one or more network nodes, and one or more servers, collectively known simply as nodes or system objects. The clients, servers, and network nodes will have 'ports' which will enable them to communicate data from one to another along a 'channel' via messages — 'request' messages will be sent from clients to servers, and 'reply' messages from servers back to clients. These 'ports' may not be explicit in the program design, however.

[1]The design has been modified in a few places to correspond more closely with how WS^3 is now

- When the system is simulated, it will proceed according to a 'virtual' time stream. The name used for the 'virtual' time units is actually irrelevant, but for consistency throughout this report, the program comments, etc., I have referred to the virtual time units as 'virtual seconds', since this is how I imagine most users will use them.

- The system will be simulated using 'process-based' rather than 'event-based' simulation. This is a conceptual difference in the implementation which will not (or at least should not) affect the answers given by a simulation. A process-based simulation models a number of processes (such as a client, a server, etc.) as concurrent threads or processes in the simulation language. A virtual time stream is maintained independently and processes voluntarily give up processing control at various points. An event-based simulation has program code which simulate events in the system, and these events schedule new copies of themselves on the virtual time stream.

  I found process-based simulation to be far more intuitive, and I was having difficulty seeing how some of things I wanted to do could be implemented using event-based simulation. Thus, I decided it would be best if I wrote the software using process-based simulation, since it was more natural to me and would thus enable me to write the software more quickly and with less errors. This seemed far more important than any small performance benefit I might gain from event-based simulation [11].

- All channels are one-way for the maximum amount of user flexibility; this allows two system objects to route messages between them in different ways, depending on which way the message is flowing.

- Servers and network nodes have queues for incoming messages, but not for outgoing messages. Clients do not have queues at all.

4.1.2 External and Interface Design Assumptions

- The program will be constructed so that it works in a batch-style mode where the user will input a specification of a system to be simulated, the program will run, and then it will detailed and summary data about the simulation. This is a more realistic project than a continuously interactive simulation, where the user could alter the system specification while the simulation was running, which might adversely affect the results.

- The program will accept an XML input file which will have a certain pre-defined structure. This input file will specify the input simulation completely (as in no order additional input would be required), and will also specify any extra parameters, optional or otherwise, that the programs requires to execute the simulation as the user wants. I decided to use XML for the following reasons:

  - It is easy for a human to read and understand it.
  - It is comparatively easy for machines to parse it, and parsing toolkits already exist for many languages.

- The program will output summary information and detailed information. The summary information will cover the entire length of the simulation's 'virtual time', whereas the detailed information will be a list of snapshots at points in 'virtual time', measuring certain parameters within the system.

- The summary information will initially be output purely to the screen in a way structured to be easy for a user to read, the detailed information will be output to a CSV (Comma Separated Variables) file so that it was easy for a program to parse. I decided upon CSV because:

  – Lots of pre-existing programs and applications have the ability to parse and import CSV data.
  – It will be comparatively easy for the program to be written to output CSV data.
  – It is easy to write a CSV parser if it is necessary to import into an analysis program which does not yet have the ability to import CSV.

4.2 Class Structure

When I was designing WS^3, it was obvious to me that I would develop WS^3 in an object-oriented language, and so I bore that in mind during my design. It is worth admitting that I was fairly sure I would develop in Java before I begun implementation, because it is a language I am very familiar with, but I made no firm commitments to a particular language during my design beyond that I would use an object-oriented language. I made the following decisions about the internal class structure of WS^3:

- Each 'system object' — each server, client, or network node — will be represented with a one-to-one mapping to an actual object in the programming language. This will make programming conceptually simple. Data private to the system object I was modelling will be stored in that object.

- I will design the system so that it used a generalised simulation toolkit. Functionality specific to the WS^3 simulation will be created by extending (inheriting from) the classes in the general simulation toolkit. Where this simulation toolkit came from was a question I decided to deal with later.

So my initial vision for the design on a very high level was essentially that shown in figure 4.1 on the next page. The classes which inherit from `WS3_SimulatedObject` each represent a type of system object and an instance of these would be created for each system object I was modelling. The `WS3_SimulatedObject` class itself is a superclass of these to support functionality common to them.

The classes prefixed with `SimToolkit` represent parts of the generalised simulation toolkit. The classes prefixed `WS3` represent classes specifically written to support features common to all simulation objects for WS^3. Hence the WS^3 class `WS3_SimulatedObject` inherits the functionality from `SimToolkit_SimulatedObject`.

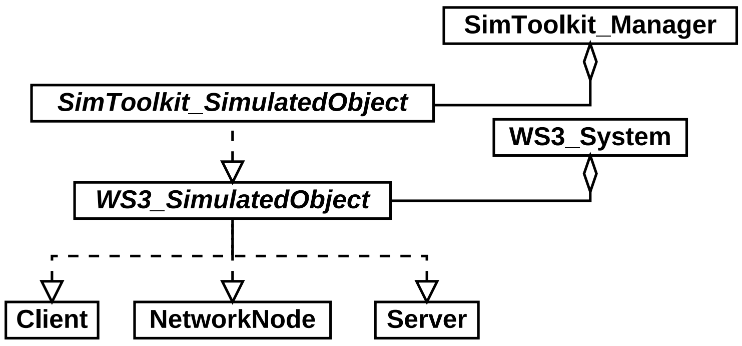

Figure 4.1: Conceptual Design of WS^3 — Revision 1

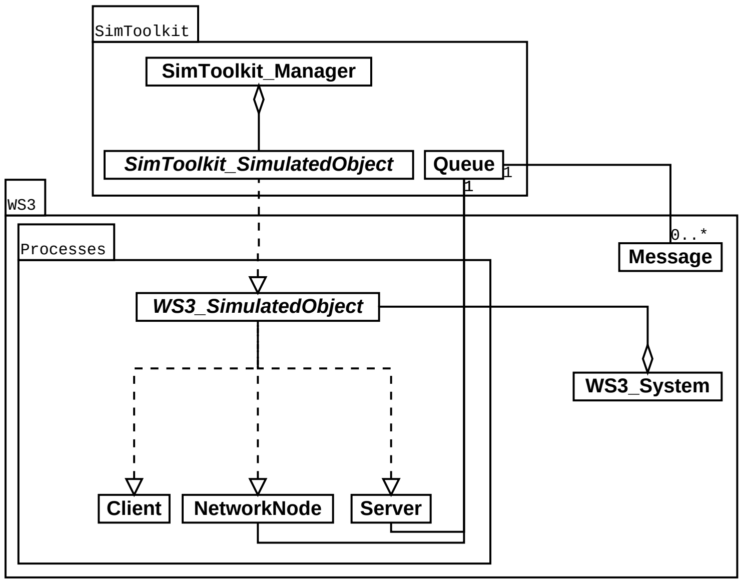

Figure 4.2: Conceptual Design of WS^3 — Revision 2

The `SimToolkit_Manager` class represents a class which has references to all the simulated objects, and would be able to start and stop the simulation code contained within them. The `WS3_System` class is responsible for a complete WS^3 simulation system and hence has references to all the relevant system objects.

I then began to refine that design, producing the class structure shown in figure 4.2. In this larger design, I began to group classes into packages to make the system easier to understand. I also added in a few classes which the system would need.

I designed one package, called `SimToolkit`, which contains the classes for the generalised simulation toolkit. This ensures that this stayed conceptually separate. Then all the other classes, which would be specific to the web simulation, are placed in the package `WS3`. Within this there is a package called `Processes`, which contains all the simulation processes — the 'active' processes which 'process-based' simulation models and which the `SimToolkit_Simulate-`

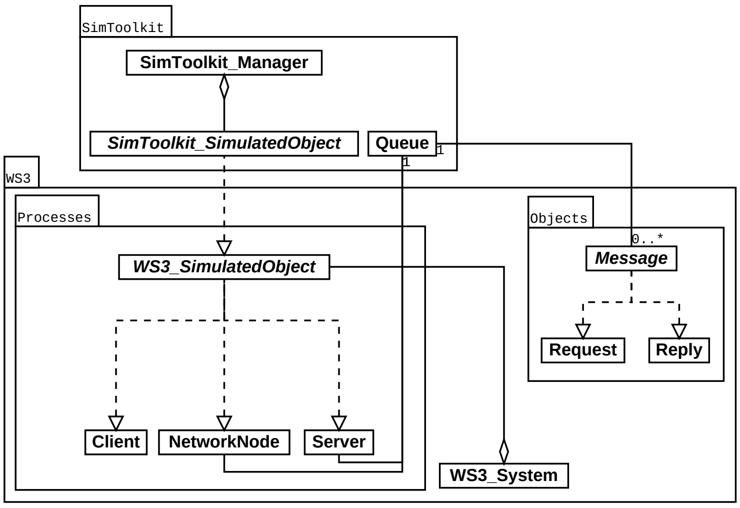

Figure 4.3: Conceptual Design of WS^3 — Revision 3

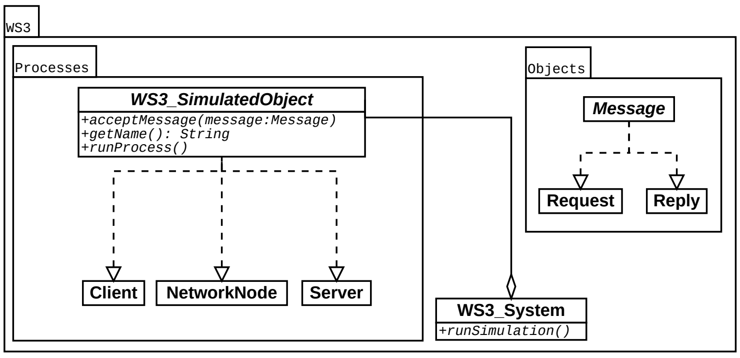

Figure 4.4: Conceptual Design of WS^3 — Revision 4

`dObject` class is the superclass of.

In the second revision, I also added a `Queue` class to the simulation toolkit. Instances of the `NetworkNode` and `Server` classes have one `Queue` each which represent their queue of incoming messages, as discussed in section 4.1.1 on page 29. The `Message` class is specifically designed for WS^3, and thus is contained within the `WS3` package. The `Message` class represents messages being sent between the system objects.

My third design revision is shown in figure 4.3. In this revision the WS^3 class `Message` is placed in a sub-package called `Objects`, made into an abstract class, and then extended with two subclasses, `Request` and `Reply`, to represent requests from clients to servers, and replies from servers to clients, respectively. This allows requests and replies to contain different information whilst treating them as similar objects.

I then started to flesh out the public interfaces of the classes I had designed. This is shown in figure 4.4.

In this revision, I concentrated upon the classes within the `ws3` package, since I was fairly sure that the simulation toolkit would be external to my project, and thus that the design of it would be mostly out of my hands anyway. Important methods are added to the `WS3_SimulatedObject` superclass to support passing messages between different objects in the system, and other actions.

I had now reached the point where I decided I had done enough 'pure' design. From previous experience I knew that there was only so much design that it was worth doing before moving on to implementation as only through implementation could I discover some of the more subtle flaws in my design. The implementation of WS^3 is described in chapter 5. If more detail on the design of WS^3 is required, please look at the source code.

Chapter 5

Implementation Issues

In this chapter I describe the implementation of WS^3.

Initially, this followed the specification described in chapter 3 and the design described in chapter 4. Some changes were made to the original design and specification, and this chapter describes the more important of these[1]. This chapter also discusses other issues that arose during implementation.

5.1 Programming Language for Implementation

The first decision I had to make, bearing in mind the design which had already occurred, was which language to implement the software in. Often this is a decision which should rightly be left fairly late, so as not to colour the design process. However, I decided fairly early on to implement the software in Java. I made this decision because:

- Java is an entirely object-oriented language, which is well suited to simulation, which requires decomposition of the system into logical objects. It is also a clean and simple language, well suited to rapid development. It contains an inbuilt threading library which helps when developing systems with multiple threads of computation, which I would be likely to use for process-based simulation.

- Tony Field [4] had already written a mostly complete, easy-to-use, and flexible simulation library in Java, which I had used during my 2nd Year Simulation and Modelling course. He granted me usage of this and the right to modify it if necessary.

- Java compilers and runtime platforms are readily available for most hardware and software platforms, thus ensuring that it would be fairly easy to run my program on different platforms if desired. Portability is normally fairly good if one sticks to using the standard Java API.

[1]Bear in mind that the specification and design in chapters 3 and 4 has been updated to reflect how WS^3 is now; so they will match with the changes as described in this chapter, not with how WS^3 was specified originally in the outsourcing report [1].

However, I did recognize that Java had some flaws for use as a simulation language. These were:

- Java is often seen as being a slow language.

  However, whilst for example it is no doubt slower in most cases for pure computation and memory manipulation than, say, compiled C, many recent advances have been made in optimizing Java compilers and JIT run-time compilers, to the extent that performance is now often similar to that seen in fully compiled languages.

  Moreover, it is now possible to compile Java source code into native code on some platforms. The tool `gcj` [42] is one example of a compiler that does this. This issue is discussed in more depth in section 8.4 on page 68.

- Java's maths libraries are not very good.

  Whilst Java's maths libraries are not as good as those in some scientific and maths-oriented languages, such as FORTRAN, it does have some maths facilities (such as those in the built-in basic data type wrapper classes, and those in `java.math`).

  Also, I had decided to use a simulation toolkit (see section 5.3 on page 38), which already contained statistical distribution samplers and the like, and hence the amount of maths-related code which I expected to have to write was minimal.

I decided that these two disadvantages were not significant enough to rule out the use of Java for the software portion of the project. As far as Java being a slow language is concerned, I decided that, in my experience, the performance penalty was minor compared with the advantage I would gain from being able to write WS^3 quickly (in my experience, Java is an easy language to program and debug). Also, as far as the maths libraries were concerned, I decided that I would have most of what I needed.

5.1.1 Input File Format

I had already decided as part of the design for WS^3 that I would use XML as the input file format. Given that I had decided to use Java as the programming language, I researched the possibilities for parsing and reading in XML files. I rapidly discovered that the best way of doing this would be to use a JAXP (Java API for XML Parsing)-compliant Java XML toolkit. After further research, I settled upon the Xerces [43] toolkit from Apache because it seemed the most stable and standards-compliant.

I decided it would be convenient to constrain the structure of an XML input file with the support of the Xerces toolkit. I did some research on this and discovered that there were two primary methods: using an XML Schema [44] or a DTD (Document Type Definition) [45]. I decided to write an XML Schema which I would supply with WS^3. This had several advantages:

- XML Schemas are easier to write than DTDs, and, unlike DTDs, are written in XML themselves. This also makes them easier to understand and learn.

- XML Schemas have a much greater range of features than DTDs. In particular, they have better constraint mechanisms. This enables me to offload a greater level of checking to the XML parser which I would be using than if I were using DTDs.
- The distribution of the XML Schema with WS^3 would enable the user to look at the XML Schema to understand the *exact* syntax of the XML input required. Thus, the file used for validation of the XML input and the user specification for it would be one and the same thing. This saves time and effort.

5.2 Changes to Specification and Design

During the implementation of WS^3, I was forced to make some changes to the specification and design for WS^3, and I chose to make others[2]. The more important of these are as follows[3]:

- I decided in general that I has relied upon use of attributes (as opposed to elements) too highly in my original specification of the XML input file. I made this decision because, upon further research, it became apparent that using attributes was inflexible in an XML document and it was harder to apply constraints on them with an XML Schema — for example, the `minOccurs` and `maxOccurs` attributes in an XML Schema could not be applied to an attribute, because an attribute could only occur a maximum of once for a given element. Thus I changed some of the attributes to elements, such as the `distribution` attribute, which became the `<creationDistribution>` and `<serviceTimeDistribution>` elements (see the user guide in appendix A for more information on the syntax of this works now).
- When I originally designed WS^3, I specified a tracing feature. However, during my implementation of WS^3, I decided to split this into two: a tracing feature and a 'data dumping' feature. The trace output from a simulation is user-readable: it consists of English statements saying what the system is doing. The data dumping output is a CSV file containing information on the state of the system over the course of the execution of the simulation. More information on the data dumping feature is in section 5.5 on page 41.
- I added the facility for WS^3 to automatically create multiple instances of a system object — this is specified in the input XML via the `instances` attribute. This made simulating large systems considerably simpler.
- I added the ability for servers to have multi-threading and multiple processors, specified via the `threads`, `processors`, and `threadGrain` attributes. I made this change so that I could study server threading.
- I added the `<queueLength>` and `<drop>` elements as valid elements in the input file. See the user guide (appendix A) for more information on these.

[2]The original specification for WS^3 can be found in the outsourcing report for this project [1]. The specification, as modified to reflect how WS^3 is now, can be found in chapter 3.
[3]For more details on how each of these impact the user, see the user guide in appendix A.

- I did not add a Poisson distribution to WS^3 as I described in the specification (chapter 3): I realised that it had been a mistake specifying this as the Poisson distribution was a discrete distribution, and hence unsuitable for a simulation where the distributions are generating random times which were not discrete.

- I realised that the 'magic' routing method that I had placed in the specification was simply unrealistic: it would be an unnecessary overhead for WS^3 to have to calculate the optimum routes for all messages, and would lead to complexity problems as the size of the system grew. Thus I decided to simply this by forcing the user to specify the routes themselves, by using the `<routeto>` element within each system object specification. This had the added advantage that the user could now specify 'unusual routes' — for example, a request could take a different path to a reply.

5.3 Java Simulation Toolkit

I re-used a simulation toolkit that had already been written by Tony Field [4] to aid in writing WS^3 because I had used it before and knew it to be flexible and that it fitted what I needed. It only allows one Thread (`SimProcess`) to execute at once, so it is thread-safe and access to member data does not have to be synchronised. This reduces performance concerns because synchronisation requires locking. It also means that software is easier to write because one does not have to be concerned with what requires synchronisation and what does not.

The class library was already well written, but I made some changes to it to increase it's suitability for what I needed:

- I added support for the Pareto distribution to the toolkit so that it could be used in WS^3, because it was drawn to my attention that it would be useful to include [22, 46, 47, 41]. Here I briefly outline how this was done:

  A density function for the Pareto distribution is[4]:

$$f(x) = \frac{a}{(1+x)^{a+1}} \text{ for } x \geq 0 \tag{5.1}$$

  To generate random variates via the inverse-transform method, this function must be integrated to produce the distribution function [47]:

  From equation 5.1,

$$F(x) = \begin{cases} 1 - \frac{1}{(1+x)^a} & \text{for } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

  We can now invert the function by setting $u = F(x)$ and solving for x [46, 47]:

[4] a is the symbol used in [47], other symbols seem to be used in different sources [48, 49, 50]. Other sources also have slightly different, though equivalent, definitions of the Pareto distribution function, which involve constant factors.

$$x = F^{-1}(u) = (1-u)^{-1/a} - 1 \qquad (5.3)$$

Hence I was able to use this result to write code to generate Pareto variates, which can be seen in the method `SimTools.Samplers.Pareto.next()`. This code simply generated a random value $u \sim U(0,1)$.

- I moved all of the packages in the simulation toolkit to my class hierarchy, according to my coding standards (as explained in section 5.7 on page 42). I did this so that my changed classes would not clash with those which Tony Field had written, if they ever happened to be installed on the same system together.

- I re-formatted the source code to fit my layout standards.

- I made a few modifications to the simulation library to increase performance, and it's coherency with the rest of my project. These changes were not significant enough to detail here, but they were documented in comments within the source code.

- Initially the simulation toolkit used double precision numbers. I was considering at one point using single precision floating point numbers for times and other time-related factors throughout WS^3, to improve it's performance, but I decided in the end to stay with the use of double precision numbers because the accuracy of the model required it and performance was not greatly affected. On average double precision numbers give about 15 digits of precision whereas single precision numbers only give about 7 digits of precision [51, 52]. I deemed that this might not be enough — for example, a simulation that ran for thousands of seconds might still need to discriminate with milliseconds of precision.

5.4 Randomness

It was drawn to my attention that, since I was using Java's standard Random number generator — that is, the random number generator in the class `java.util.Random` — I should research the methods used by this generator to produce it's random numbers to see if they were 'random' enough. Specifically, it was important to ensure that the period of the random numbers was long enough. Any pseudo-random number generator has a 'period' after which random numbers repeat themselves — this is a by-product of the process and occurs because the same random seed will always produce the same random numbers, given the same method [53].

I discovered that the contract of `java.util.Random` was not very tight. It specified that the class should generate random numbers according to a linear congruential formula, a typical method, and refers to the source [53]. A linear congruential method uses the generative formula:

$$x_{n+1} \equiv \lambda x_n + \mu \pmod{2^\beta} \qquad (5.4)$$

(where x_n is the 'previous' random value, x_{n+1} is the current value, and λ and μ are constants. β is the word size on the machine in question)

However, as [53] makes clear, the period of the random number generator depends upon what values one uses the constants in the formula (λ and μ), and these were not specified in the contract for `java.util.Random`. Theoretically, then, I could not rely upon any particular implementation correctly providing a sufficiently long random period or providing sufficiently 'random' numbers (what this constitutes is beyond the scope of this report — see [54] for more information). As [55] states, many ANSI C implementations provide a "totally botched" implementation of linear congruential random number generators.

Initially, thus, I thought it would be sensible to write my own random number generator to ensure that I knew the values of λ and μ, and thus that they were valid. Indeed, I did write a 'wrapper' class around `java.util.Random`, in preparation for this.

However, I did some more research [56, 53, 54, 25], and it quickly became apparent that this was also quite a naïve view. The problem is that floating point arithmetic works subtly differently on different machines. Although in theory the Java virtual machine abstracts away a lot of that detail, in practice some machine-dependent eccentricities still exhibit themselves, and thus the selection of λ and μ should be dependent on the machine on which the simulation is running [56].

Thus I decided that the best idea was to leave the implementation of the random number generator up to the Java API after all — thus any machine dependent behaviour could be dealt with by the implementor of the Java runtime on that architecture. However, I left my random number wrapper class in place in case it became useful in future. In fact, I did use it to ensure that the random seed used by WS^3 is always 1 — this ensures that results are exactly repeatable.

Nevertheless, I decided it would increase confidence in WS^3 if I wrote a utility to test the random numbers on a particular Java implementation. Thus I wrote the `doc.ajf98.websim.RandomTest` utility, which is supplied with WS^3 (for more information on how to use `RandomTest`, see sub-section A.6.3 on page 77). It works on the assumption that a poor linear congruential random number generator will fairly quickly work back to it's initial value (since all random number generators have a finite period). Thus it generates a stream of random numbers, and if one occurs which matches the first random number it generated, it aborts, reporting the total number of random numbers generated. If this number is low (i.e. not of the order 2^{64}, which is approximately the number of different `double` values Java can represent [57]), the user knows that the implementation of the random number generator on their system is poor. This program can be run for as long as necessary to convince the user that WS^3 will have a good quality period of random numbers to work with.

For the record, I have run the random number generator test successfully on the following software up to the values specified:

- The Blackdown Java 1.3.1 SDK on Debian GNU/Linux [58] — up to $\sim 24 \times 10^9$ unique random numbers. Took approximately 6 hours.
- IBM Java JDK 1.3.0 on SuSE Linux [59] — generated $\sim 744 \times 10^9$ unique random numbers. Took approximately 36 hours.
- Sun Java JDK 1.3.1 on Windows 2000 — generated $\sim 1.1 \times 10^9$ unique random numbers. Ran for approximately 1 hour.

5.5 Equilibrium

Another issue which was drawn to my attention whilst I was writing WS^3 was that of equilibrium. In most simulations, it is important to understand the role of equilibrium.

Equilibrium essentially means that as a simulation progresses — as the virtual time tends towards infinity — the statistical distributions which represent the state of the system become constant. This does not mean that the state of the system *is* constant, merely that it becomes 'stable' and 'hovers' around particular values. In the case of WS^3, for example, this might mean that the length of the network node and server queues become stable, and that the server utilisations also become stable. This means they do not grow or shrink over a long period of time.

It sometimes important to understand whether a system is in equilibrium. For example, if one had a simple web system with one client and one server, and the client was capable of generating requests at a far greater rate than the server was capable of processing them, the server queue length would grow without limit over time (assuming an infinite queue length). Thus the mean queue length over the entire length of the system would be an almost meaningless statistic. This is because the system would never have been at equilibrium.

In order that the user of WS^3 could decide whether the system was at equilibrium, I implemented a data dumping feature. This output certain data about the objects in the system during the execution of the simulation, at user-specified discrete intervals of virtual time. This data is output in CSV format, which allows the user to important into a large variety of spreadsheets and analysis tools. The data can then be plotted, and the moment at which the system reaches equilibrium can be seen. The simulation can be re-run using the `resetStatsPoint` attribute (which is applied to the `<system>` element). This will run the simulation, and reset the statistics at the specified number of virtual seconds, which ensures measurements are only done from the equilibrium point onwards.

However, it is important to appreciate that it is possible for a system not to reach a meaningful equilibrium. For example, it is quite likely that a bank's online web banking system will never reach a state of equilibrium, because the demand on the system will clearly vary over the day as different number of people (clients) want to access it. On an even higher level, the system might not reach an equilibrium over a week or even a year because the demand will be continually varying according to the day of the week, holidays, and so on. Also, the system might reach a transient equilibrium for a while, then, due to a change in the demand, shift to another mode entirely.

This latter point is not directly relevant to WS^3 because WS^3 as implemented does not have the facility to vary the demand over time: clients' demand always follows the same statistical distribution. However, it is important to understand that if WS^3 were extended in this way, that the question of whether the simulation were at equilibrium might become less important. This kind of extension is discussed further in sub-section 8.4.1 on page 68.

5.6 Documentation

When writing WS^3, I produced two kinds of documentation:

- A user guide. This can be found in appendix A.
- Javadoc comments [60] and standard Java comments. I placed Javadoc comments in my code to document the interfaces of various classes within WS^3. I also placed normal Java comments in certain places to document certain non-obvious implementation features. However, I have always guarded against placing too many comments in source code, as I feel that making the source code self-explanatory is more important, so by no means all of the source code is completely documented.

5.7 Standards

During my development, I adhered to the following standards:

- In general, I complied with Sun's document 'Code Conventions for the Java Programming Language' [61] as far as possible.
- I created all my classes in packages with the prefix `doc.ajf98`. I considered that this was sufficiently unique to ensure that any classes I had written would not clash with any that others had written. There is an ad-hoc convention to use the reverse of a domain name for packaging classes, which would have led to me using the prefix `uk.ac.ic.doc.ajf98` or similar. However, I deemed that this was too long and cumbersome.

  The simulation classes that I used from Tony Field were migrated into my package hierarchy — i.e. that beginning with `doc.ajf98`. This was so that they would not clash with any future classes he might decide to write or modify.

Chapter 6

Testing

This chapter explains the testing that I did to WS^3, which is based upon the testing specification in my outsourcing report for this project [1]. Most of the testing was done via a test and evaluation suite which can be run automatically across WS^3. I split this test suite into two halves — TST and CNC.

This chapter describes the TST part of the test and evaluation suite. The TST input files were designed to test the correctness of all of the features of WS^3. It describes each of the test inputs that I developed, what they were designed to test, and explains any problems with WS^3 that they drew to my attention. Section 6.8 on page 49) describes tests that did that did not involve the input file suite.

The test cases in this chapter are not separated into categories, as some tests are designed to test more than one aspect of WS^3. However, as a group, they are designed to test all of the features of WS^3. *Not all of the tests carried out are documented in this chapter* in order to ensure conciseness; however, a representative sample is included[1].

The CNC part of the suite was used to evaluate some hypothetical web serving models with WS^3, and is discussed in chapter 7.

6.1 TST-001

This is the most basic test in the testing suite: it simulates a client and a server, directly connected. It is designed to test a basic simulation with WS^3, which can be verified with queueing theory. It also tests WS^3's tracing capability — tracing is turned on to the maximum level. The setup is shown in figure 6.1 on the next page[2].

The expected results are:

$$\begin{aligned} \text{Server utilisation} &\approx 1 \\ \text{Average client response time} &\approx 1 \end{aligned}$$

[1]For more details on running the test suite, see section A.6 on page 76.

[2]A quick guide to the network diagrams within this chapter can be found in appendix B on page 89.

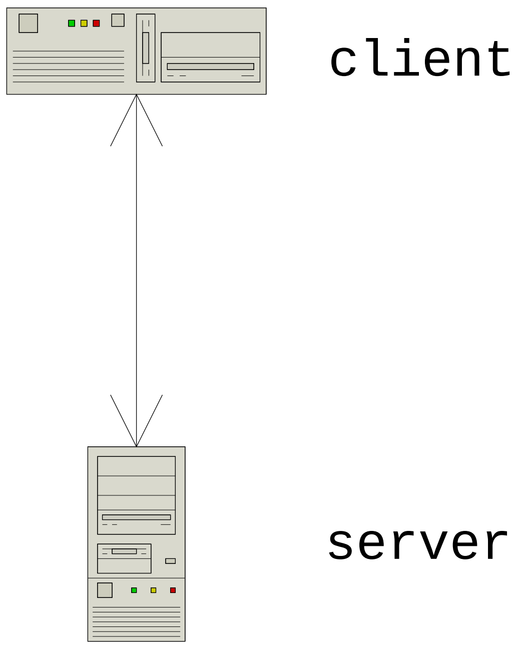

Figure 6.1: TST-001

We can show this using queueing theory. The test is a fairly simple M/M/1 queueing model[3]. A single client and a single server were used, and the client inter-generation time was determined by an exponential distribution, as was the service time on the server, both with a rate of one event per virtual second.

Hence:

$$\lambda = \mu = 1 \tag{6.1}$$

(λ is the mean arrival rate and that μ is the mean service rate — for more information on queueing theory, see [12]).

Hence:

$$\text{Server Utilisation} = 1 - P(0) = 1 - (1 - \varrho) = \varrho = \frac{\lambda}{\mu} \tag{6.2}$$

($P(0)$ is the probability that there are no requests in the system, ϱ is the traffic intensity, which is measured in Erlangs).

Hence, I can conclude that the server utilisation should be 1.

This test has always produced the expected results since I wrote it. Also, the trace file produced is similar to that shown in figure 6.2 on the next page, which is correct and complete.

Other tests similar to TST-001 which used queueing theory to validate WS^3 were also carried out, but are not detailed here for conciseness.

[3] M/M/1 is written in Kendall notation, a notation used to describe queueing models — see the glossary for more information and other sources

```
Connection from client[0] : Created to object server[?].
Potential destination for client[0] : Created as server...
Client client[0] : Created.
Routing map for client[0] : Being formed.
Routing map for client[0] : Adding element with target:...
Connection from server[0] : Created to object client[?]...
Server server[0] : Created.
Routing map for server[0] : Being formed.
Routing map for server[0] : Adding element with target:...
Server Thread server[0](0) : Created.
Simulation system  TST-001 :  created with maximum runti...
@0.0 Server server[0] : Looking at queue.
@0.0 Server server[0] : Going to sleep.
@1.0 Client client[0] : Created {Request for /testFile_3...
@1.0 Object client[0] : Attempting to find route for mes...
@1.0 Client client[0] : Route found. Sending {Request fo...
@1.0 Server server[0] : Asked to process message {Reques...
@1.0 Server server[0] : Looking at queue.
@1.0 Server server[0] : Handing over message {Request fo...
@1.0 SThread server[0](0) : Has been asked to process {R...
@1.0 Server server[0] : Going to sleep.
@1.0 SThread server[0](0) : Entering execution loop.
@1.0 SThread server[0](0) : Waiting to get control of a ...
@1.0 SThread server[0](0) : Has control of a processor. ...

...

@1.8 SThread server[0](0) : Has control of a processor. ...
@2.0 Client client[0] : Created {Request for /testFile_2...
@2.0 Object client[0] : Attempting to find route for mes...
@2.0 Client client[0] : Route found. Sending {Request fo...
@2.0 Server server[0] : Asked to process message {Reques...
```

Figure 6.2: Example trace output from TST-001 (edited to fit)

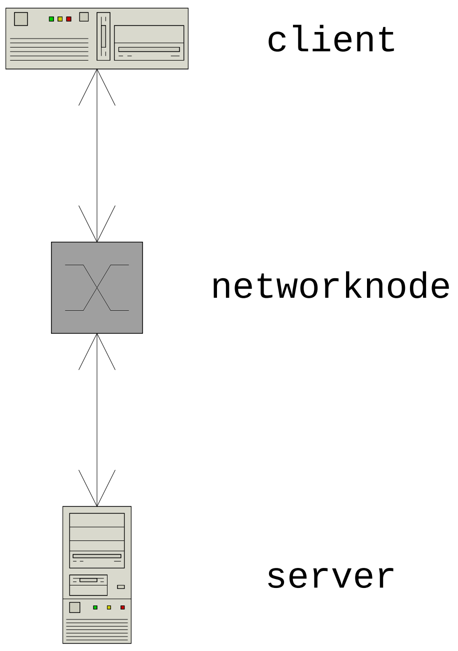

Figure 6.3: TST-002

6.2 TST-002

This test is designed to simulate a fairly basic system, with one client and one server, but unlike TST-001, it also has an intermediate network node. The layout can be seen in figure 6.3.

This test helped me discover the following bug:

- I wrote the test incorrectly at first, and missed out a connection that the system needed. WS^3 was missing a check for whether there was a return connection in the `findRoute` method of the `RouteableSystemObject` class. I corrected WS^3, and corrected the test.

Apart from that bug, this test has always correctly executed under WS^3.

6.3 TST-003

This test is designed to stress-test the WS^3 XML loading mechanism. I created this test because I found a bug in WS^3 whereby some object names, which I had intended for it to accept, it did not accept. Thus I created this test which contains lots of boundary conditions and unusual values and identifiers, to ensure that WS^3 does not crash in these circumstances. There is no diagram for this test because it has long and complex object names.

Since I created this test, it has never failed to execute correctly — WS^3 has always loaded it, and executed it to completion.

```
<system>
</system
```

Figure 6.4: TST-005

6.4 TST-004

This test is designed to test the multiple-threading, multiple-processor features of WS^3, by specifying a simple client-server system (similar to that shown in figure 6.1 on page 44 for TST-001). The test specifies four concurrent threads, three processors available, and a custom thread grain of 0.4 virtual seconds.

Since I created this test, WS^3 has always loaded it and executed it to completion correctly, with valid results.

6.5 TST-005

This test is an exact replica of the test from figure 5.1 in the outsourcing report [1], which is designed to be invalid XML. The objective of the test is to ensure that WS^3 does not crash under such circumstances, but aborts with an error instead. The test is shown in figure 6.4.

The output from WS^3 if this test is executed is shown in figure 6.5. The parser indicates a number of errors relating to the format of the input file, but WS^3 does not crash.

This test has been used to refine the error output from WS^3, but has never caused WS^3 to crash.

```
WSSS (WS^3) x.x.x.x
Copyright (C) Andrew Ferrier (andrew@new-destiny.co.uk) 2002.
http://www.new-destiny.co.uk/andrew/project/

Initializing system.
DocumentBuilderFactory is ignoring comments: true
DocumentBuilderFactory is ignoring element content whitespace: true
DocumentBuilder is namespace aware: true
DocumentBuilder is validating: true
DOMImplementation supports 'Core': true
DOMImplementation supports 'XML': true
DOMImplementation supports 'Traversal': true
ERROR: Problem parsing input file on line 5, column 9:
Element type "system" must be declared.
ERROR: Problem parsing input file on line 8, column 1:
The element type "system" must be terminated by the
    matching end-tag "</system>".
ERROR: General file parsing error: Stopping after
fatal error: The element type "system" must be terminated
by the matching end-tag "</system>"..
```

Figure 6.5: Output from WS^3 on execution of TST-005 (edited to fit)

```
<fruit>
    <banana>
    </banana>
</fruit>
```

Figure 6.6: TST-006

6.6 TST-006

This test is an exact replica of the test from figure 5.2 in the outsourcing report [1], which is designed to be valid XML, but invalid for WS^3 according to the WS^3 schema. XML. The objective of the test is the same as for TST-005: to ensure that WS^3 does not crash under such circumstances, but aborts with an error instead. The test is shown in figure 6.6.

The output from WS^3 if this test is executed is shown in figure 6.5. This is correct: the parser indicates that the input elements are incorrect and WS^3 aborts, but does not crash.

This test has been used to refine the error output from WS^3, but has never caused WS^3 to crash.

```
WSSS (WS^3) 0.3.14.7
Copyright (C) Andrew Ferrier (andrew@new-destiny.co.uk) 2002.
http://www.new-destiny.co.uk/andrew/project/

Initializing system.
DocumentBuilderFactory is ignoring comments: true
DocumentBuilderFactory is ignoring element content whitespace: true
DocumentBuilder is namespace aware: true
DocumentBuilder is validating: true
DOMImplementation supports 'Core': true
DOMImplementation supports 'XML': true
DOMImplementation supports 'Traversal': true
ERROR: Problem parsing input file on line 5, column 8:
Element type "fruit" must be declared.
ERROR: Problem parsing input file on line 6, column 13:
Element type "banana" must be declared.
```

Figure 6.7: Output from WS^3 on execution of TST-006 (edited to fit)

6.7 TST-007 and TST-008

These two tests together fully test all the distributions available in WS^3 and also test many other features: they are quite complex. A diagram, which represents either TST-007 and TST-008, is shown in figure 6.8 on the facing page. I also used them to white-box test WS^3: the tracing is turned on to the maximum level in both. What the final results of the simulations were was not of interest to me, since I knew it would be unsolvable for me via queueing theory and thus

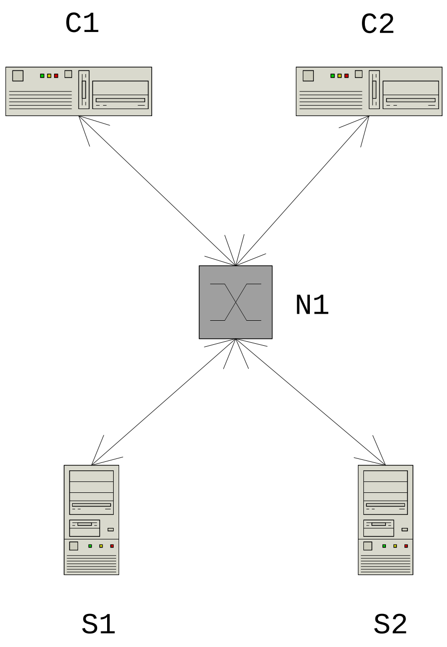

Figure 6.8: TST-007 or TST-008

I could not check them. However, I examined carefully the trace output and checked some of the results by hand, to ensure that the simulation was executing correctly.

WS^3 has never failed to execute these tests correctly.

6.8 Non-File-Suite Testing

I also did testing with WS^3 whilst I was implementing it, where the test files were not part of the TST testing suite — this testing was part of the normal implementation routine. Some of the more important of these tests are explained here:

- More tests were done to ensure the correctness of WS^3's simulation: some of these involved verification with queueing theory, some more informally via 'rules of thumb'. I haven't detailed these in this chapter for conciseness.
- A requirement for WS^3 was that all of the CSV output of WS^3 (i.e. the data dumping output), should be syntactically valid[4]. I could not perform an explicit test on this, but I used the data dumping output a lot whilst I was writing WS^3 and using the TST and CNC testing and evaluation suite and I never once encountered a problem importing CSV output into an application.
- Another one of the requirements which I had set for WS^3 in the specification in my outsourcing report was that WS^3 should never crash. To

[4]This was specified in the testing specification in my outsourcing report [1].

a certain extent, some of the TST tests tested this, but also, whenever I caused WS^3 to crash during development, I documented it and made sure that the problem was tracked down and fixed. Thus I am reasonably confident that WS^3 is stable and will not crash easily — though it is likely that there is a fatal bug somewhere, as with most complex software!

- I decided *not* to test WS^3 with a tool such as Fuzz [62] as I had suggested in my outsourcing report because any bugs so found would most likely be in the XML parser I was using (Apache Xerces) rather than in WS^3 itself, since the parser was responsible for parsing of the input file.

The next chapter (chapter 7) evaluates WS^3 by, amongst other things, looking at the CNC input file suite, which complements the TST input file suite.

Part III

Uses for the Software

Chapter 7

Evaluation

This chapter evaluates WS^3 quantitatively and qualitatively by: using the CNC input file suite[1] to evaluate some hypothetical web serving models; examining the speed of WS^3's simulation; and discussing the accuracy of WS^3's simulation.

7.1 Notes About the Input Files

Only summary information is included about the CNC input files in this chapter. For more detail, see the sections in the user guide on the files provided and the test suite (section A.5 on page 76 and section A.6 on page 76). All input files in this chapter were equilibrium-adjusted where appropriate (as explained in section 5.5).

7.2 Web Serving Guidelines

7.2.1 Size of Machines Used

One of the first things I wanted to do when evaluating WS^3 was address one of the example 'general' questions which I posed in the introduction (chapter 1):

> Whether it is best, generally, to use many small interconnected serving machines (a web server 'farm'), or to use one or only a few large serving machines.

It seemed to me, that taking the question above, there were two extremes — where there was only a single high-powered serving machine, and where there was a very large number of serving machines, nominally with the same total processing capacity. I decided to model the first in the evaluation input file `CNC-001.xml` and the second in `CNC-002.xml`.

[1]See the introduction to chapter 6 for more information on the TST and CNC input file suite

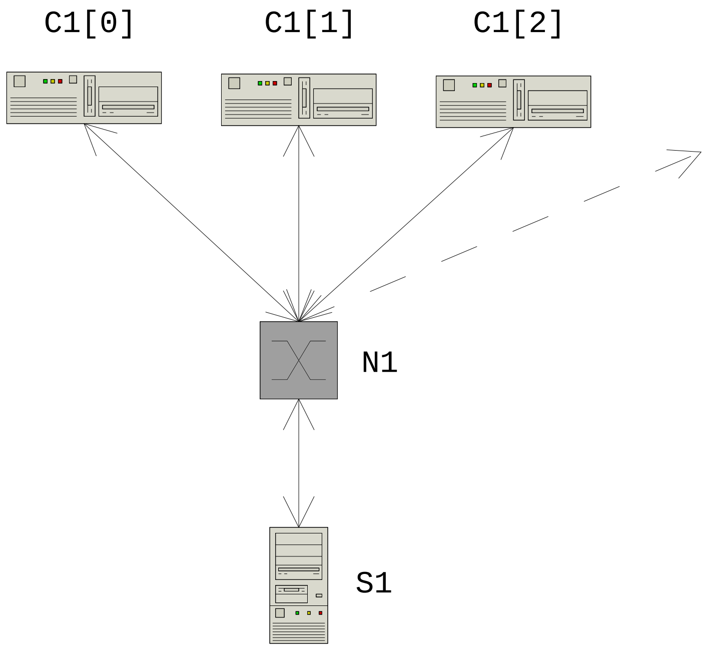

Figure 7.1: CNC-001

$C1$	Creation Distribution	Exponential	rate=0.2
$C1$	Number of Instances	50	
$S1$	Service Distribution	Exponential	rate=100

Table 7.1: Initial parameters for CNC-001

CNC-001

CNC-001 simulates a network with 100 clients ($C1[0]$ .. $C1[99]$), which are linked to a network node ($N1$), which is in turn linked to a single server ($S1$). This is really the simplest sensible network for analysing this problem.

The layout for CNC-001 is shown in figure 7.1. The dashed arrow indicates that there are more than 3 clients, but that they are not all shown on the diagram.

Next, I created CNC-002 which was the 'opposite' of CNC-001 to test whether, for a given set of parameters, the server utilisation and the client response time increased or decreased. However, before I created CNC-002, I calibrated CNC-001 so that the server utilisation was at a certain nominal value — I picked 50%. To do this, I first picked some values that I thought would be sensible, which are shown in table 7.1.

I then adjusted the client creation distribution until the server utilisation was about 50% (in fact, it was 0.499). The final parameters are shown in table 7.2 on the next page.

$C1$	Creation Distribution	Exponential	rate = 1
$C1$	Number of Instances	50	
$S1$	Service Distribution	Exponential	rate = 100

Table 7.2: Final parameters for CNC-001

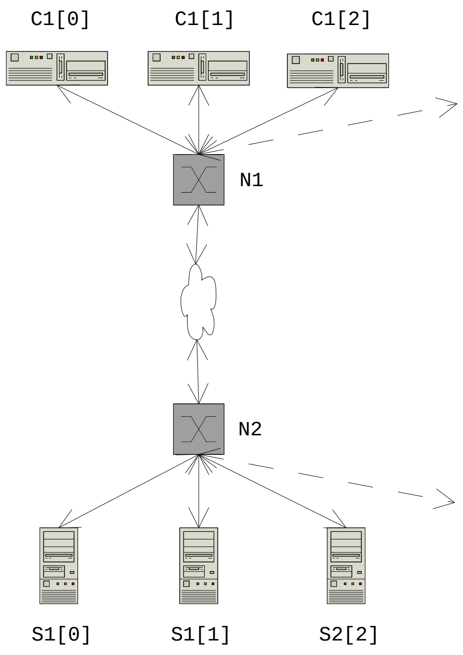

Figure 7.2: CNC-002

CNC-002

Once CNC-001 was constructed, I constructed CNC-002, basing it around CNC-001. However, I increased the number of servers to a large number (20). I then reduced the exponential service rate for each of the servers to $\frac{100}{20} = 2.5$, ensuring that overall, the servers had nominally the same power. The layout for CNC-002 is shown in figure 7.2. The dashed line at the bottom indicates that there are more than 3 servers, but that this is not shown on the diagram.

When I executed CNC-002, the results were almost exactly the same as for CNC-001. The server utilisation for each of the servers was approximately 50%. I decided I could conclude from this one of two things:

- Using several ‘small’ servers versus one ‘large’ server was approximately of the same efficiency, and thus the decision on which to use should be a cost-based decision, or:
- My model was not sufficiently complex to fully model the subtleties of the problem.

It would be naïve to think that there was not a certain amount of truth in the second explanation; after all, the simulation model which WS^3 uses is never going to model all the complexities of a web server system. However, there is also likely to be a certain amount of truth in the first explanation too. Certainly it fits with intuition. And if there is truth in it, it leads to an interesting conclusion: that because one server is (fairly obviously) easier to administer than multiple servers, then it is better to use one large server (ignoring the problems of server failure). Certainly this conclusion would merit further investigation; unfortunately this would require a more in-depth model than I had time to include in WS^3. Thus I decided that it was not worth me investigating this question further given the complexity of WS^3, and the simulation model which it allowed.

7.2.2 Server Threading

WS^3 has the feature to simulate multiple threads on a processor. I used this feature to try to answer the second of the two 'general' questions that I posed in the introduction (chapter 1):

> How many synchronous threads should exist on a single server system for maximum performance.

I first approached this by creating a variety of input files. CNC-003 was the first I created, and was identical to CNC-001. I then created a number of copies of this file, and altered the number of threads and processors on the server. Since the exponential service rate for CNC-003 was 100, my approach was to divide this by the number of processors available. This would mean that all the processors aggregated would have nominally the same power as the single processor machine (CNC-003). The parameters and the server utilisations are shown in table 7.3 on the next page.

I created a larger variety of samples than that shown in table 7.3, and I tried analysing them in a variety of ways, plotting different graphs, and trying to come up with a pattern. Below is presented the conclusion.

As can be seen from table 7.3, the server utilisations for most of the inputs were ~ 1, which was to be expected considering that the servers were all nominally of the same power.

The most striking patterns were to be found when looking at the client response times to the server requests. A graph of the client response times is shown in figure 7.3 on page 58 (for processors = 40 only). As can be seen, the client response time is at a minimum when:

$$\frac{Processors}{Threads} \approx 0.75$$

However, the graph has two peaks, and the pattern which the response time follows is not at all obvious. This is most curious, and it seems quite likely that the pattern which the response times are following is quite complex. There are obviously certain ratios of processors to threads which work well for the clients and certain ratios which do not. The complex pattern seen in figure 7.3 is reminiscent of the patterns created when one sums several sinusoidal waves of different periods with each other. There is a high likelihood that there are lots of interacting factors which are causing the complex patterns which one sees in figure 7.3, and that sometimes these factors contribute to each other, and

Name	$S1$ Threads	$S1$ Processors	$S1$ Exponential Service Rate	$S1$ Measured Utilisation	$C1$ Average Response Time
CNC-003	1	1	100	0.50	$\sim$ 10
CNC-008	20	10	10	0.99	$\sim$ 12.3
CNC-014	20	15	6.67	0.99	$\sim$ 4.5
CNC-004	20	20	5	1	$\sim$ 3.5
CNC-015	40	10	10	0.997	$\sim$ 5.7
CNC-021	40	15	6.67	0.99	$\sim$ 12.37
CNC-025	40	17	5.82	0.99	$\sim$ 8.8
CNC-009	40	20	5	0.99	$\sim$ 8.1
CNC-026	40	22	4.54	1	$\sim$ 20.8
CNC-022	40	25	4	0.999	$\sim$ 7.1
CNC-027	40	28	3.57	1	$\sim$ 7.9
CNC-023	40	30	3.33	1	$\sim$ 4.75
CNC-028	40	32	3.12	1	$\sim$ 6.8
CNC-024	40	35	2.86	1	$\sim$ 8.3
CNC-007	40	40	2.5	1	$\sim$ 5.4

Table 7.3: Measurements for differing numbers of threads and processors

sometimes they cancel out the effects of each other, depending on the processor to thread ratio. It is unlikely in practice that one would be able to discover these factors analytically, particularly since real systems are likely to be more complex than CNC-003, and thus the conclusion is that one should use software like WS^3 to model the situation one intends to use to discover the best number of threads to use.

7.2.3 Distributional Differences

As well as using WS^3 to answer some of the questions I had posed in the introduction, I also decided to investigate some other interesting features of WS^3 and simulations.

One of the first things I wanted to do was analyse the difference between different statistical distributions.

I did this by creating a simple system, which I called CNC-050, whereby client requests were created at a constant rate of 1 per virtual second, and servers had constant service times of 1 request per virtual second, and then created a few variations on this where the server distribution was of different types (though with the same mean inter-service time). After executing CNC-050, the server utilisation was 1, as expected, and the mean client response time was 1 virtual second.

The layout of the basic system is shown in diagram 7.4 on the next page.

Below I outline what I discovered for each variation:

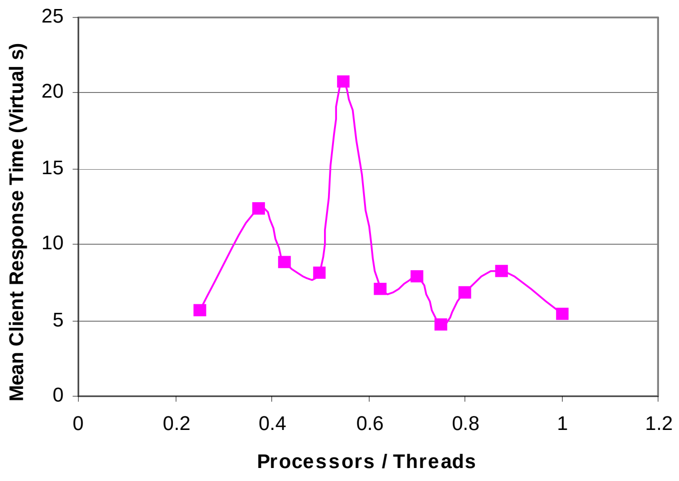

Figure 7.3: Client response times for different number of threads, processors = 40

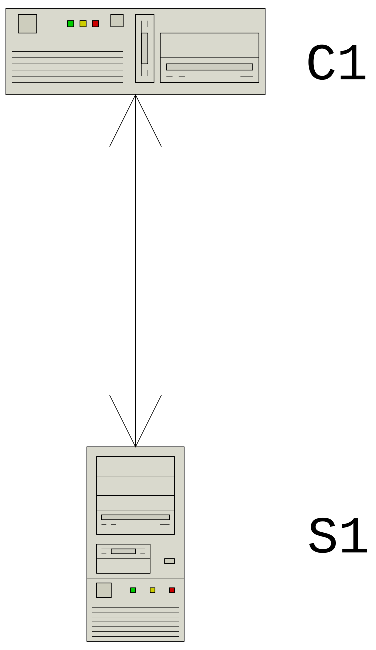

Figure 7.4: CNC-050 et al.

CNC-051 — Exponential Distribution

This was a variation on CNC-050 where an exponential distribution was used in place of the constant distribution on the server. The rate specified for the exponential distribution was 1, and I expected this to provide approximately the same results as for CNC-050, since the mean of the exponential distribution is the same as it's parameter (the rate). And, indeed, one of the results was as expected — the server utilisation was ~ 1. However, the mean client response time was ~ 60, and I did not expect this — this was a factor of 60 greater than before!

I took a closer look at the summary statistics from the simulation, and explained the difference this way: the client also had a very high variance in the response time. Before, in CNC-050, the variance in the response time had been 0, as would be expected, because constant distributions were used throughout. Now the variance was ~ 530. Here, a distribution was being used on the server which varied greatly the time taken to service a request. Since the variance in the client response time was considerably greater than the mean, I could conclude that the high value of the mean was misleading.

CNC-052 — Uniform Distribution

CNC-052 was a variation on CNC-050 which used a uniform distribution which was distributed between a lower bound of 0.5 and an upper bound of 1.5. The mean of this distribution is $\frac{0.5+1.5}{2} = 1$, which is the same as that for CNC-050. Thus I was expecting similar results as for CNC-050, and indeed they were very similar. Thus I concluded that the uniform distribution was similar in simulation behaviour to the constant one that I used in CNC-050 because the results were similar. This is borne out by common sense: the uniform distribution is very simple, and over time will tend to produce results similar to a constant distribution if the means of both are equal.

Pareto Distribution

The Pareto distribution is discussed in sub-section 7.2.4.

7.2.4 Real-world Example

Unfortunately I did not have a lot of time during my project to try to apply WS^3 to many specific real-world examples.

However, one example I tried was modelling the DoC webserver. I wrote a program which analysed the DoC web server logs for the period of one week and determined the average and variance of the interarrival times of requests. This was done with the help of some log parsing software that converted web server logs into an easier-to-use form [63].

Using this method, I discovered that the average inter-arrival time for requests on the webserver was $0.75s$, and that the variance of the inter-arrival time for these requests was $\sim 2 \times 10^6$.

Thus I created the simulation specification CNC-030, represented in figure 7.5 on the next page. I gave it two sets of clients, $C1$ and $C2$. $C1$ represented the clients local to DoC, and $C2$ represented, other, more remote, clients. $C1$ clients only had to go through network node $N1$ to get to the server, $S1$,

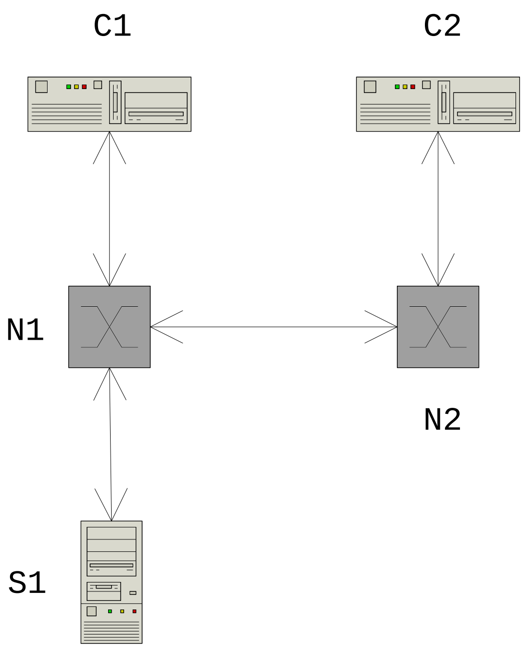

Figure 7.5: CNC-030

but $C2$ clients had to go through both network nodes ($N1$ and $N2$). Network node $N2$ was given a higher fixed service delay than $N2$. Server $S1$ was given a very high service rate with 8 threads, which is a common default for the Apache webserver that DoC uses. The creation distributions on $C1$ and $C2$ were set to emulate the inter-arrival rates above, using (initially) a normal distribution.

Thus this all matched the DoC web serving configuration as closely as I could given the limited amount of information at my disposal.

The results were as follows:

The average $C1$ (local) client response time was ~ 0.1 seconds. The average $C2$ (remote) client response time was lengthier, at about ~ 0.2 seconds. The network node utilisations were negligible (near to zero). The server utilisation was about 0.06. All of this matched up roughly with what one would expect: clients within DoC retrieve web pages near-instantly. Clients further away get a good response time, but it depends upon their network connection (I did not model different remote clients in CNC-030). The server utilisation is very low, which likely matches that in DoC (although my model did not address dynamically created pages, because WS^3 does not have that feature — see subsection 8.4.1 on page 68 for more information — which would probably lift the server utilisation dramatically).

I tried to alter this example to utilise a Pareto distribution for the client inter-request generation rate, since this would be likely to more realistically model the client requests [41]. However, the simulation failed to approach equilibrium (equilibrium is explained in section 5.5 on page 41), and I left it executing for several days. Thus I could not conclude anything about how the Pareto distribution would have affected this simulation, except that it is very difficult to cause simulations to converge to equilibrium when they use the Pareto distribution,

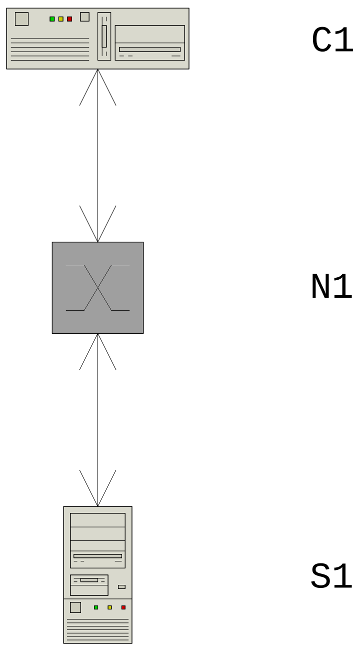

Figure 7.6: CNC-060

which was something I had already learnt [64].

7.3 WS^3's Speed

Another thing I decided it would be interesting to analyse was the speed of WS^3 when the input simulation specification altered. Optimising WS^3 for speed was not one of the primary objectives of the project, but I thought it would be interesting to try to come to some conclusions about what kinds of simulations tended to affect the speed.

One of the things I had noticed whilst developing and using WS^3 was that the larger the number of system objects that were present in the simulation, the longer the simulation tended to run. I decided to test this theorem.

I created a system (which I called CNC-060), which was a simple system with a single client, a single network node, and a single server. The layout of the system is shown in diagram 7.6. All of the system objects were initially given an exponential distribution with a rate of 1. I then created some variations on this where there were a greater number of clients, with all the clients attached to the network node. I increased the exponential rates at the network node and the server accordingly so that they could 'handle' the load that was being placed upon them.

I then ran all of these input tests on an unloaded real machine for $10,000$ virtual seconds. The results are shown in table 7.4 on the next page and are plotted on the graph shown in figure 7.7 on the following page.

As can be seen from the graph, there appears to be an approximately linear relationship between the number of system objects and the total runtime

Name	Number of Clients	Real Runtime
CNC-060	1	27s
CNC-061	2	39s
CNC-062	3	50s
CNC-063	4	64s
CNC-064	5	74s
CNC-065	6	86s
CNC-066	7	98s
CNC-067	8	110s

Table 7.4: WS3's Speed for varying numbers of clients

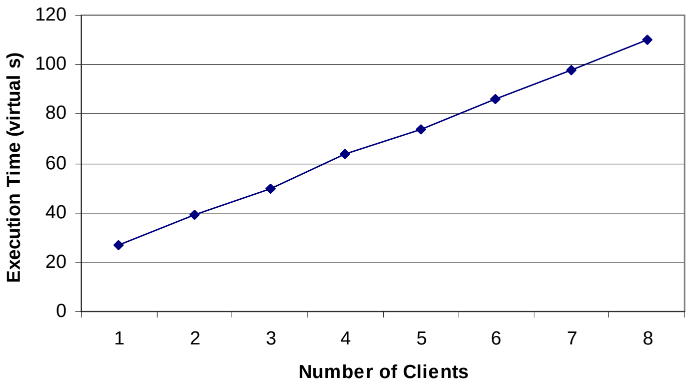

Figure 7.7: Runtime for varying numbers of clients

provided the other parameters are kept the same. This can be explained in the following way: when there is a greater number of system objects, WS^3 has to context-switch between a greater number of threads (which is proportional to the number of system objects) in order to 'cover' the same period in virtual time. Thus it is quite logical that there is a linear relationship.

I did not investigate the speed difference imposed on WS^3 by the use of different statistical distributions, because I decided that this would not be 'fair' — some distributions take longer to generate a variate than others, particularly those that use the rejection method [25], and this would have an impact on the simulation speed, whilst providing no difference in functionality. Nevertheless, it might be an interesting future project to investigate this, whilst attempting to remain 'fair'.

7.4 WS^3's Accuracy

Another one of the things that I felt is was necessary to discuss in the context of WS^3 was it's accuracy. I have already discussed equilibrium issues and how they affect WS^3 in section 5.5 on page 41.

However, it is important to understand that simulation results can vary, dependent upon the random values used and the length for which the simulation is run. Investigating this problem further was unfortunately not something that I time to spend doing with WS^3, but it would be important if one were using WS^3 in a real-world case — there are standard methods for doing output analysis [11], which could be used. It must be remembered, however, that these tend mainly to rely on empirical rules-of-thumb [41].

Chapter 8

Conclusions

This chapter summarises my project, how I executed it, what I've achieved, and what WS^3 has been useful for. It also discusses future possibilities for WS^3 and web server simulation and analysis.

As stated in the introduction to this report, my project had two primary aims:

- To construct the web serving simulation software — WS^3.
- To use this software to come to some general guidelines for web serving systems.

Both of these aims were furthered with this project as I explain below.

8.1 Construction of WS^3

I produced a piece of software, WS^3, which exceeded the requirements I had initially laid out in the outsourcing report for this project [1]. The final version of WS^3 has the following features:

- It supports simulation of clients, network nodes, and servers via an input specification of a system, coded in XML.
- Connections can be specified between system objects.
- Routing tables can be specified for system objects, so that messages can be sent along user-defined routes.
- Clients can create requests according to a large variety of statistical distributions.
- Service times for network nodes and clients also vary according to one of the same distributions.
- It is possible to specify which servers a particular client sends requests to.
- Network nodes can be set to randomly 'drop' messages.

- Servers can have an arbitrary number of synchronous software threads, and an arbitrary number of processors. The context switching between threads on a processor is simulated.
- All system objects (clients, network nodes, and servers) can be easily replicated up to arbitrary numbers. Connections and routes are also automatically replicated where appropriate.
- Server and network node queue lengths can be fixed at a particular value or made infinitely long.
- Summary and detailed data dump information is output.
- Tracing of actions in the simulation can also be output.

WS^3 was tested for validity (as explained in chapter 6).

The construction of WS^3 has been very useful. WS^3 would need more development if it were to be used as a professional server sizing tool — in particular, it would need customisation such that it matched the particular serving software and hardware in use more accurately, because at the moment it abstracts away too much detail from such things. However, it is interesting to use at the moment, simply to understand how such systems work. In particular, reading the trace output from WS^3 gives a very good idea of the way serving systems behave. And because all of the source code for WS^3 is publicly available, as well as this report, which gives a good overview of the way WS^3 works, it could easily be customised to many particular situations.

8.2 General Conclusions about Web Systems and WS^3

Using WS^3, I came to the following general conclusions (which are described in more detail in chapter 7):

- The ratio between the number of threads and the number of processors on a server has an important and complex impact on the mean response times seen by clients, which is very difficult to analyse without using simulation software such as WS^3.
- At a high level, lots of low-powered servers or one high-powered server seem to be approximately equivalent; hence, due to the clear maintenance advantage of having a low number of servers, that is the preferential option. However, this would require more investigation for detailed test cases.
- The use of different statistical distributions with simulations such as WS^3 can produce unexpected results; one must be careful to use a distribution which is appropriate to the system one is modelling.
- The speed of simulation software such as WS^3 tends to roughly proportional to the number of process objects being simulated, but is also affected by the statistical distributions in use.
- It can be difficult for simulations which use the Pareto distribution to converge; but it is useful for modelling real-world client demands.

8.3 Strengths and Weaknesses of WS^3 and the Project

This project was, as has been acknowledged by many people, quite ambitious. Given this, I feel it has been quite successful. Some of the strengths and weaknesses of the project were:

8.3.1 Strengths

- WS^3 is a functioning piece of software, which, to my knowledge, is relatively free of bugs, and has a good feature set. With a bit more development I believe it could be used for practical server sizing problems.

- WS^3 is written in an easy-to-maintain style and in a language which I believe is well suited to it; Java. It also uses XML as the input format and again I believe that this has worked well. If I wrote WS^3 again I would use Java and XML again.

- I have come up with some interesting guidelines for server sizing which are interesting in themselves and also merit further investigation.

- Although WS^3 has some web-specific features, it could also be fairly easily used to model many types of client-server systems, with appropriate modifications to the simulation code. Examples include EJB (Enterprise Java Beans)-based application servers [65], email systems, and so on.

8.3.2 Weaknesses

Below I explain some of the weaknesses of WS^3 and the project; these can be taken as comments on how I would have liked to improve the project, if I did it again.

- Because WS^3 was a simulator, using it to discover general guidelines and relationships is difficult; In retrospect, I should probably have given more emphasis to analysis in the project. Nevertheless, I believe that WS^3 is a useful tool and that I have come up with some useful conclusions. Ideas for further analysis are examined in sub-section 8.4.2 on page 69.

- WS^3 does not directly support an 'empirical' distribution where data from real-life can be used as a distribution in the simulation model: one has to take empirical data and fit it to one of the standard distributions supplied with WS^3. If I had more time to spend on WS^3, it is one of the first things I would add, because it makes input modelling (matching the input to real styles of input) considerably easier.

  The reason it was not done is because it is requires an interface to WS^3 to input the data, as well as a method of sampling the data from the distribution: these would require specifying in the input file. Thus it would, in order to have the level of flexibility that would be useful and which I wanted, be quite complex.

- I would have liked to have spent more time validating the general accuracy of WS^3's simulations, although I am reasonably confident from the testing, both formal and informal, that I did, that the simulation models the system as intended.

- There are also other features I would liked to have added to WS^3; see section 8.4. The lack of these is partly a weakness but they are also an opportunity for future development.

8.4 Future Extensions

Unfortunately there simply was not enough time available for me to do all the things I would have liked to do with my project — there are features I would have liked to add to WS^3, and more analyses I would have liked to carry out. I feel that it is important to document these, so some of the more of these extensions are listed below. I very much enjoyed my project, and I intend to continue developing WS^3, so some of these may well be carried out in the future.

8.4.1 WS^3 Features

Modelling Features

- WS^3 does not support time-based demand variation. Most web servers do not have the same load on them all the time. It's fairly obvious that, for example, in the middle of night, most web servers have a considerably lower load on them than they do during the middle of the day. It would be good if WS^3 supported clients which alter their demands over time. However, this would definitely be a non-trivial feature to add in.

- As mentioned above in sub-section 8.3.2 on the page before, WS^3 does not support an 'empirical' distribution. This is a high priority to add to WS^3 in the future.

- It might be useful in some situations if WS^3 supported termination based on some other criterion than just the 'current' virtual time — for example, the total number of client request/replies. However, I decided that most practical simulation could be done with WS^3 the way it stood, and that the simulation could always be re-run with a different running time as appropriate. Thus I didn't implement this.

- WS^3 does not support any concept of 'big' and 'small' requests — i.e. all requests which arrive at a server have the same service time distribution. This would be an interesting feature to add, where different requests were serviced in a different ways, possibly also with the idea of pages being dynamically formed, maybe with data from a back-end database.

- WS^3 has no support for the concept of client 'timeouts'. As it stands at the moment, clients will wait for a request to be returned indefinitely. This is unrealistic as most client software in common use will timeout after a certain period of time: a minute for example. However, this only becomes relevant if the system is failing, and what's more, WS^3 will still currently indicate, for example, if the average reply time for a client request is 10 minutes, which is clearly too long, and would indicate a problem.

- There is currently no way for the number of threads on a server to change dynamically (say, for example, according to the load on the server). This would be an interesting feature to build in, because some real-life web servers behave in this way.

Implementation Changes

- I would have like to use the `java.util.TimerTask` for the thread that notifies that there is still progress in the simulation. This would probably make WS^3 more efficient, because this thread would automatically wake up at the appropriate time, and would not need to be checked at any other time.

- Currently, the `SimulationSystem` class does not support multiple instances of itself being created without bugs exhibiting themselves. I would like to improve this so that, for example, multiple simulation systems could be specified in a single input file and created concurrently.

Other Features

- I would like to spend more time using optimising and native code compilers, as well as directly optimising the code, to make WS^3 itself faster. One example of this was an idea I had to speed up the `Queue` object by using a pointer to the current head of the queue and a rotating buffer, rather than creating and destroying objects, which is expensive in most Java implementations. Another example would be using the `gcj` Java native-code compiler on Linux. But this was quite an extensive area of investigation and it does not actually affect the simulation results themselves, merely the execution time of WS^3.

8.4.2 Analysis of Web Serving

- I would like spend more time on analysing real-world examples with WS^3. Unfortunately, in the time available, it was difficult to get hold of real-world data and map it accurately onto a simulation with WS^3. Nevertheless, if I were to do the project again, this is something I would like to focus upon to a greater extent. In particular, I would liked to have studied the impact of non-finite queue lengths. WS^3 has the ability to simulate these, but I did not have time to do any analysis with them.

- I would like to spend more time doing queueing theory-based analysis to check and evaluate WS^3. As it stands, the queueing theory which I used was reasonably basic — it would be interesting to use queueing theory with some more complex models to validate WS^3's validity in that context.

Part IV

Appendices

Appendix A

User Guide

This user guide is also available as a standalone document, primarily for external users. The program source code (and compiled binaries), as well as all reports for the project can be found at [1].

This user guide assumes a fairly technical reader, who does not require a lot of guidance in installing software etc.

A.1 Licencing

WS^3 is subject to the licence conditions in the file `LICENCE`, supplied with all pre-packaged versions of WS^3. Apache Xerces, [43], also supplied with all pre-packaged versions of WS^3, is subject to it's own licence conditions, found in the file `LICENCE.xerces`.

```
This product includes software developed by the
Apache Software Foundation (http://www.apache.org/).
```

A.2 An Overview of WS^3

WS^3 emulates a web serving system. It does this by reading in an XML input file that you specify, parsing it, and constructing a simulation of that system in memory that represents the system you have specified in the input file. It then executes that simulation system for a period of time that you have specified, using various parameters that you have specified. When the simulation is finished, WS^3 will output various data on the state of the simulation. Data can also optionally be output during the execution of the simulation.

There are three main objects which can exist in a WS^3 simulation system — clients, servers, and network nodes. There must be at least one client and one server, but network nodes are optional. Objects communicate by sending messages to each other along connections — these messages are either requests (sent by clients to servers) or replies (sent back from servers to clients in response to requests). Network nodes, if used, are used purely as intermediaries. Routes are used to specify ways of getting from one system object to another if they are not directly connected with a single connection.

Network nodes and servers have incoming message queues. Incoming messages are added to the queues and remain there until all the items ahead of them have been processed. For network nodes, processing merely involves sending on the message via an appropriate route. For servers, processing involves parsing the message (which should be a request), formulating a reply message and sending the reply message back via an appropriate route.

Clients do not have incoming message queues. Replies are processed immediately. Reply processing does not involve anything except recording the fact that the reply has been received.

A.3 System Requirements

- An operating system which can support the Java 2 Environment, such as Windows 2000 or Linux. Other operating systems also support the Java 2 Environment and WS^3 should operate correctly with those too, but it has not been tested with them. Installation instructions in this user guide are only given for Windows and Linux.

- A Java Run-Time Environment which is compatible with the Java Software Development Kit, version 1.3.1, as provided by Sun [13], such as the Java Run-Time Environment 1.3.1. *This environment must be correctly installed as per the instructions that come with it, such that the* PATH *and* CLASSPATH *environment variables are set correctly.*

  Note: If you want to build WS^3 from source code, you should have the full Java Development Kit installed rather than just a run-time environment.

- A Java XML Parser with the following features:

  - Support for XML 1.0 [14].
  - Support for DOM (Document Object Model) Level 2.
  - Support for JAXP.

  An example of a parser with those features is the Xerces Java Parser, version 1.4.4 [43]. This has been tested with WS^3 far more extensively than any other Parser. Some other parsers may cause problems when used with WS^3. In particular, the Apache Crimson Parser has been briefly tested and I had some problems with it, as it does not support XML Schema validation.

  The Xerces Java Parser, version 1.4.4 is supplied with all pre-packaged versions of WS^3. It is subject to it's own licencing conditions, found in the file `LICENCE.xerces`. It is recommended that you use this version of Xerces as it has been tested with WS^3.

- If you wish to build WS^3 from the source code using the supplied makefile, you must have GNU (GNU's Not Unix) `make` [66] installed, as well as a standard suite of Unix utilities, especially `xargs`. On Linux or other Unix systems, this is often already installed. On Windows the easiest way to obtain this functionality is to install a Unix emulation layer such as Cygwin [67].

A.4 Installing WS^3

WS^3 will come distributed in one of the three following types of files:

- A `.zip` file.
- A `.tar.bz2` file.
- A `.tar.gz` file.

A `.zip` file should be used if you are planning to install on Windows. If you plan to install on Unix/Linux, use the `.tar.bz2` or `.tar.gz` file. Which you use will depend on what decompression programs are installed on your system — `gz` decompression is more widely available, but the `bz2` file will be smaller to download.

A.4.1 Installing on Unix/Linux

Note: The following commands should be entered from a shell prompt.

Uncompress the archive into a temporary directory, then change to that directory. For these instructions, we are going to assume that the directory is `/ws3`. For example, if using the `.tar.bz2` file[1]:

```
mkdir ~/ws3
tar xvjf ws3.tar.bz2 --directory=~/ws3
cd ~/ws3
```

Or if using the `.tar.gz` file:

```
mkdir ~/ws3
tar xvzf ws3.tar.gz --directory=~/ws3
cd ~/ws3
```

The archive is distributed with a pre-made binary `.jar` file that you can use to run WS^3 (after installing it). However, if you want to re-build WS^3, execute the following commands:

```
make clean
make all
```

In order to install WS^3, use the following command:

```
make install
```

A.4.2 Installing on Windows

Unzip the `.zip` file (archive) into a directory somewhere on your computer, for example `c:\ws3`.

[1]The `j` option is not available on tar on some systems. Type `man tar` or `man bzip2` to find out how to decompress the file

A.4.3 Warning About Installation on any Platform

Note: It is important to ensure that however you install WS^3, the `xerces.jar` file is in the same directory as `ws3.jar`, *or* that `xerces.jar` is in your Java Runtime Environment's official extension directory. No other location will be likely to work, *even* if that location is in your `CLASSPATH` — this is because WS^3 is invoked with the `java -jar` option, which ignores the `CLASSPATH`. For more information, see [68]. The standard `Makefile` supplied with WS^3 will install both of these files in the same place, so this problem should not occur unless you try to install manually.

A.5 File Structure Once Installed

Once WS^3 is installed, the following directory and file structure will exist in the directory you have installed into[2]:

```
doc/         - The Java source files and compiled classes
tests/       - The input test files
ws3.jar      - The WS3 Jar file which can be used to invoke WS3
xerces.jar   - The supplied Apache Xerces Parser
ws3.xsd      - The WS3 XML schema
README       - Miscellaneous information
LICENCE.*    - Licence information
Makefile     - The makefile which can be used to rebuild WS3
```

A.6 Testing the Install of WS^3

A.6.1 Testing on Unix/Linux

Bring up a shell prompt and change to the directory where you extracted WS^3 (e.g. ~/ws3). Execute the following command:

```
make test
```

A.6.2 Testing on Windows

Bring up a command prompt in Windows. From the directory where you extracted WS^3 (which, if you followed the instructions above, will be `c:\ws3`), and execute the following command:

```
make test_noinstall
```

WS^3 will build and install itself if necessary, then run through a variety of test simulation specifications designed to ensure that it is installed correctly. These test simulation specifications are the TST and CNC files referred to in the Testing and Evaluation chapters of the final report of this project [1]. Bear in mind that some of the tests are designed to fail!

[2]Only important files and directories are shown.

A.6.3 Randomisation Test

WS^3 utilises the random number generator supplied with your Java Runtime Environment as a source of randomness. You can run a test to ensure that this random number generator does not ‘repeat’ itself too quickly. From the directory where you installed WS^3, type:

```
make randomtest
```

A utility will load and begin generating random numbers. It will take many years to complete execution, unless your random number generation implementation is faulty, in which case it may abort fairly quickly with a warning. You will want to abort it after a while, when you are confident than it has run for longer than you will ever want to use WS^3 for. Press Ctrl-C to do this.

See the final report for this project [1] for more information on these randomness issues.

A.7 Rebuilding the Javadoc Documentation

The WS^3 archive file is supplied with Javadoc documentation pre-built in the `docs/` directory. If you wish to rebuild this documentation, enter the following command from the directory where you unpacked the archive:

```
make doc
```

A.8 Creating an Input File

An input file for WS^3 is an XML file, which conforms to a certain XML Schema. This schema is supplied with WS^3 (the file supplied is called `ws3.xsd`), and specifies what elements can occur in the XML file, in what order, with what content, etc. — in other words, it constrains the content of the XML to a subset of valid XML. If you are not familiar with XML or XML schemas, it is recommended you use an XML editor which supports schemas to create any and all XML files which are input to the program, such as XML Spy [69]. WS^3 checks the XML file is valid for the schema[3], but the error messages thus produced directly reference the content of the schema and thus may be counter-intuitive to users not familiar with how schemas work. The XML file should conform to the basic layout shown in figure A.1 on the following page. This will ensure that the program correctly uses the right schema file, and is well-formed XML.

A.8.1 Points to be Noted about Input Files

The following points should be noted about input files to WS^3:

- Identifiers are used to specify names and refer to system objects in the input file. Identifiers can contain only the alphanumeric characters (upper and lower case), and the underscore (`_`) and dash (`-`) characters, and must

[3] In most cases, assuming one is using a schema-validating XML parser library, such as Xerces 1.4.4.

```
<?xml version="1.0"?>

<system xmlns="http://www.new-destiny.co.uk/andrew/project/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        dataDumpPeriod="dataDumpPeriodInVirtualSeconds">
    <name>inputSystemName</name>
    <runtime>runtimeInVirtualSeconds</runtime>

    <!-- a number of system object elements -->

</system>
```

Figure A.1: An outline layout for how an input XML file to WS^3 should look
.

be between 1 and 127 characters long. All identifiers are case-sensitive. Being case-sensitive means, for example, that `clientA`, `ClientA`, `CLIENTA`, and `Clienta` are all names of different system objects. However, one should try to avoid using different system objects which are differentiated only by case in that way, as it will most likely serve only to confuse, although WS^3 will understand the difference.

- All time units are generally assumed in this guide to be 'virtual seconds', or in terms of 'virtual seconds'. It actually makes no difference to WS^3 what one assumes the time units to be, but I generally refer to them as virtual seconds as that is a convenient time unit which seems appropriate in most cases. However, some aspects of the system will have 'sensible' default values if they are not specified in the input file, and these default values are based around the assumption that the virtual time units are seconds. Also, whatever time unit one uses, one *must* be consistent throughout an input file or WS^3 will not produce the desired results.

- Since a WS^3 input file is merely an XML file, XML-style comments can be used anywhere that they are normally valid in an XML file. Comments start with the string `<!--` and end with `-->` and they can span multiple lines. An example has already been used in figure A.1.

A.8.2 Input File Elements

A well-formed WS^3 XML input file consists of a single `<system>` element, with various attributes and subelements.

The `<system>` element *must* contain `xmlns` and `xmlns:xsi` attributes exactly as they are in figure A.1. There are also various non-compulsory attributes of the `<system>` element:

- `defaultTTL` — The time-to-live value that is initially assigned to each message in the system. This is an integer greater than zero. The default value, if the attribute is not specified, is 32. See section A.8.6 on page 85 for more information on time-to-live values.

- `dataDumpPeriod` — Specifies how often in virtual seconds WS^3 dumps out data. It should be an integer greater than zero. The default value, if the attribute is not specified, is 1. Data dumping is described in more detail in sub-section A.9.1 on page 86.

- `traceLevel` — Specifies the level of tracing which WS^3 will use when running this simulation. This should be a integer which ranges between 0 (no tracing) and 4 (full tracing). A number higher than 4 will produce nothing extra beyond full tracing. A number lower than 0 will cause an error. See subsection A.9.2 on page 86 for more information on tracing.

- `resetStatsPoint` — If present, specifies the virtual time point at which the summary statistics will be reset. This helps the user with obtaining accurate statistics about the execution, whilst compensating for equilibrium. For more information on this, see the section on 'Equilibrium' in the project final report [1]. Must be a number of virtual seconds greater than zero. If it is absent, or equal to, or less than, zero, the statistics will not be reset.

There are various sub-elements of the `<system>` element. Some are compulsory and some are optional. They are as follows, and whether they appear or not, *must* appear in this order:

- `<name>` — Specifies the name of input system. This name is not currently used for anything by WS^3 except as a title etc. when outputting information. It must appear once and only once.

- `<runtime>` — Specifies the total system runtime in virtual seconds. It has no default and must appear once and only once.

- `<client>`, `<networknode>`, `<server>` — These elements specify clients, network nodes, and servers in the simulation system. These three elements have various attributes and sub-elements. They are similar for each of three elements so they will be described together in sub-section A.8.3. There must be at least one `<client>` element and one `<server>` element in the system — `<networknode>` elements are entirely optional.

A.8.3 System Objects

This sub-section describes attributes and subelements for the `<client>`, `<networknode>`, and `<server>` elements, collectively known as the system object elements.

There are various non-compulsory attributes for these system object elements. They are:

- `instances` — Specifies how many instances of a system object are created. This should be an integer greater than zero. The default value if the attribute is not specified is 1. If more than 1 instance is created, then WS^3 automatically creates multiple system objects with the same name and will represent those objects by appending `[`n`]` to the end of the name, where n is the instance number of the system object (numbering starts from 0). When connections are created to an object, they should still be created just to the base name of that object, even if there are multiple instances of that object. Messages will be sent to a random object instance.

- `threads` — Valid only for the `<server>` system object. Specifies the number of concurrent threads operating on the modelled server. Must be an integer greater than zero. The default value if the attribute is not specifies is 1. If one sets this attribute, one normally wants to set the `processors` attribute as well.

- `processors` — Specifies the number of processors available on this server. The number of threads executing concurrently can never exceed this number. Must be an integer greater than zero. The default value if the attribute is not specifies is 1.

- `threadGrain` — Specifies the time grain which threads execute for. The length of time for which one thread can execute without giving up control to another thread can never exceed this value. Must be a decimal number of virtual seconds greater than zero. By default this attribute is set to 0.1 virtual seconds.

There are also various subelements for the system object elements. Some of them are compulsory and some are not. They are as follows, and must appear in this order if they appear:

- `<name>` — Specifies the name of the system object. This must comply with the identifier guidelines discussed in section A.8.1 on page 77. This element must occur once and only once.

- `<connectto>`, `<routeto>` — These elements specify connections from, and routes for, the system object. At least one connection must be specified and zero or more routes can be specified. Connections and routes are explained in more detail in section A.8.4 on the facing page.

- `<creationDistribution>` — Valid only for the `<client>` element. Specifies the distribution used for creating the inter-generation times for client requests. The distribution is specified using a single distribution subelement which is one of the elements described in section A.8.5 on page 83.

- `<serviceTimeDistribution>` — Valid only for the `<networknode>` and `<server>` elements. Specifies the distribution used for creating the service times for network node queues and server queues. The distribution is specified using a single distribution sub-element which is one of the elements described in section A.8.5 on page 83.

- `<destPossibility>` — Valid only for the `<client>` element. Specifies a server which it is valid for the client to create requests for. In other words, when a request is created by a client, WS^3 picks a random destination possibility from all those specified by that client (with equal probability). If only one destination possibility is specified for a client, the client always generates messages that are destined for that destination possibility. At least one `<destPossibility>` subelement must be specified for each `<client>` element.

  Note: The decision on which server to send a message to is made independently and before any decisions on how to route the message based on `<connectto>` and `<routeto>` elements (which are explained in more detail in section A.8.4 on the next page).

- `<queueLength>` — Valid only for the `<networknode>` and `<server>` elements. Specify the length of the incoming message queue (not including the message currently being processed). Can be either a integer value greater than zero or the special value `infinite`, which ensures that queue is of infinite length. Obviously this is a modelling aspect which would not exist in the real world, but it is useful if one is unsure what the maximum queue length should be or one wishes to model a system in such a way that one can relate it to queueing theory easily. This element must appear once and only once.

- `<drop>` — Valid only for the `<networknode>` element. The value of the element must be a number between 0 and 1. Represents the probability that the network node should 'drop' any given message which it is asked to process.

A.8.4 Connections and Routes

Each system object must have at least one outgoing 'connection' (although one can have as many as desired) and has zero or more outgoing 'routes'. A connection models a physical connection between two system objects — for example, if one were modelling a simple system with two PC (Personal Computer)s connected directly to the same hub on a simple LAN, one acting as a web server (call it $S1$), and the other as a web client (call it $C1$), then one would probably model the link between these two with one 'connection' in WS^3 (though other models are possible which would be equally valid). Connections are *always* one-way, so in this case one would have two 'connection' objects — one attached to the client, going to the server, and one attached to the server, going to the client.

A route allows one to build more complex systems. If one had only two system objects, a route would be useless, but if there are more than two system objects, routes allow one to specify ways of getting from one system object to another, via another third party system object (or possibly more than one third party), even when there is no physical 'connection' between the two.

For example, say one extended the previous example so that there were two client hosts (call them $C1$ and $C2$) on the LAN, with the hub now modelled separately as a network node object ($N1$), and still with one server ($S1$). This example is shown in figure A.2 on the following page.

Ignoring the return journey from the server to the client to keep things simple, the way one would typically model the physical 'connections' for this in WS^3 is:

- There is a physical 'connection' from $C1$ to $N1$.
- Similarly, there is a physical 'connection' from $C2$ to $N1$.
- There is also a physical 'connection' from $N1$ to $S1$.

However, this is inadequate for WS^3 as it stands: if $C1$ generated a request to be sent to $S1$, WS^3 would not know the route it should take: all it knows are the targets of the *immediate* neighbours that it has *physical* 'connections' to. Hence we need to create some 'routes':

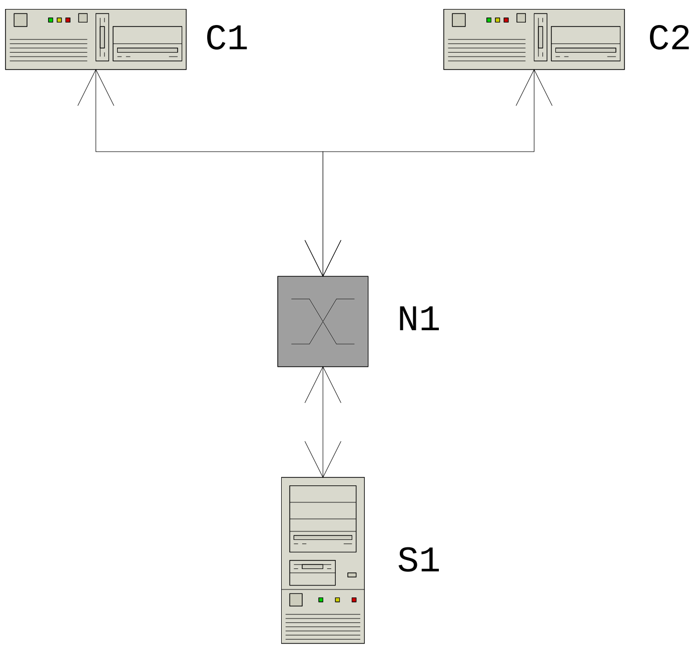

Figure A.2: A simple example of a web client-server network

- Create a route at $C1$ where the destination is $S1$ and the route to take is via $N1$.
- Similarly, create a route at $C2$ where the destination is $S1$ and the route to take is via $N1$.

This would solve the problem: when $C1$ created a request to be sent to $S1$, for example, WS^3 would have a route whose ultimate target is $S1$, even though $C1$ does not have a physical connection to $S1$. Thus WS^3 would first sent the request to $N1$ where it would be queued on $N1$'s incoming queue. When $N1$ got round to dealing with the request, it would know how to send it to $S1$ because it has a physical connection to get it there. If there was no physical connection, $N1$ could sent the packet on again if it had a route which had the target $S1$ (although in our simple example it did not).

If this is confusing, think of connections and routes in the following way:

- A connection is attached to a source system object and has one parameter:
  - The name of the target system object.
- A route is attached to a source system object and has two parameters:
  - The *destination*, which is the *ultimate* target for messages.
  - The *route*, which is the name of an object the source object has a connection to — the next 'hop' which will bring the message 'closer' to the target.

Connections and routes are created between system objects by placing `<connectto>` and `<routeto>` elements inside the appropriate system objects in the input file. It is easiest to show this with an example (we will use the same setup as before, with a four-system-object network). The partial specification is shown in A.3 on the next page.

A.8.5 Distribution Types

WS^3 supports a variety of statistical distributions which can be used for specifying client request generation, network node service time, and server service time. Each `<creationDistribution>` and `<serviceTimeDistribution>` element must contain a specification for one, and one only, distribution. One of the following distribution elements can be chosen:

- `<constant>` — This is a very simple distribution, that returns the same value every time it is used. For example, if one wants to model a client which generates a new request every 10 seconds, one would specify the `<creationDistribution>` of the client as in figure A.4 on page 85.

- `<exponential>` — Defines an exponential distribution, which follows the following formula:

  $$X = -\rho \times \ln r \tag{A.1}$$

  where X is the variate, ρ is the parameter, and r is the randomly generated value.

  The single parameter is the average rate for the distribution. For example, if the XML in figure A.5 on page 85 were used in a `<client>` element, then the client would create 10 requests every virtual second.

- `<uniform>` — This element defines a uniform distribution, which returns values that are distributed uniformly over the interval specified. For example, if one wants a client to generate requests uniformly between very 3 seconds and every 5 seconds, one would specify the `<creationDistribution>` of the client as in figure A.6 on page 86.

- `<geometric>`, `<erlang>`, `<positiveNormal>`, `<weibull>`, `<pareto>` — These distributions are all used in a very similar way to the distributions above. For the sake of conciseness, I have not detailed how each works, but they are all standard statistical distributions and there is plenty of information available for each one [25, 11, 20]. The Geometric and Pareto distributions only have one parameter each, which is included directly in the distribution element (similar to that in figure A.4 on page 85). The other three extra distributions use the parameters shown in table A.1 on page 85.

WS^3 does not actually support an 'instantaneous' distribution, where messages are routed instantly. However, one can be emulated, for most practical purposes, by creating a `<constant>` distribution and setting the routing time to be a value considerably smaller than any other time period used in the simulation specification. Bear in mind that although this works, it can slow down the simulation considerably.

```
<client>
    <name>C1</name>
    <connectto>N1</connectto>
    <routeto>
        <destination>S1</destination>
        <route>N1</route>
    </routeto>

    <!-- other specification elements -->

</client>
<client>
    <name>C2</name>
    <connectto>N1</connectto>
    <routeto>
        <destination>S1</destination>
        <route>N1</route>
    </routeto>

    <!-- other specification elements -->

</client>
<networknode>
    <name>N1</name>
    <connectto>S1</connectto>

    <!-- other specification elements: the network node needs
         no routes as it has a direct connection to S2 and
         we are not concerned about the return journey in
         this example -->

</networknode>
<server>
    <name>S1</name>

    <!-- other specification elements: we have specified no
         connection for the server as we are not
         concerned about the return journey in this
         example: but this would not be adequate for
         a real WS3 input file as every system
         object must have at least one outgoing
         connection -->

</server>
```

Figure A.3: An example of how to use the `<connectto>` and `<routeto>` elements

```
<creationDistribution>
    <constant>10</constant>
</creationDistribution>
```

Figure A.4: An example use of the `<constant>` distribution.

```
<creationDistribution>
    <exponential>10</exponential>
</creationDistribution>
```

Figure A.5: An example use of the `<exponential>` distribution.

A.8.6 Time-to-live Expiry

All messages have a TTL (Time-to-Live) value. For each message, it starts off at 32, unless you specify a different value for the `defaultTTL` attribute (which is attached to the `<system>` element). Each time a message is routed by a network node, the TTL value is decremented. If it reaches zero, then the message is discarded (though WS^3 produces a warning and the drop is recorded).

A.8.7 Constraint Enforcement

The XML Schema that is supplied with WS^3 enforces a number of constraints on the input files which WS^3 will accept. Most of these are common sense and they are fully specified in the XML Schema file itself, so they will not be fully detailed here. However, a quick summary of some of the more important ones follows:

- Item names must be consistent across the input file. For example, one cannot specify a `<connectto>` element which connects to a system object that does not exist.
- Item names must conform to the identifier guidelines found in subsection A.8.1 on page 77.

A.9 Running WS^3

To run WS^3, simply issue the following command[4]:

[4]The `../` preceding the schema filename is necessary due to a bug in Java. You will always need to adjust this so that is points to the schema in the directory above the directory the schema is actually in.

Distribution Name	Parameter 1	Parameter 2
Erlang	`<k>`	`<theta>`
Positive Normal	`<mu>`	`<sigma>`
Weibull	`<alpha>`	`<beta>`

Table A.1: Parameters for statistical distributions in WS^3

```
    <creationDistribution>
        <uniform>
            <lbound>3</lbound>
            <ubound>5</ubound>
        </uniform>
    </creationDistribution>
```

Figure A.6: An example use of the `<uniform>` distribution.

```
java -jar ws3.jar inputfile.xml ../ws3.xsd
```

It is possible that this command may not execute correctly if another version of the Xerces parsing classes are installed on the system you are using, as is the case in DoC for example. In this case, you will have to execute WS^3 via the compiled class file rather than via the `jar` file [5]. Execute the following command from the directory where you installed WS^3 [6]:

```
java -classpath .:xerces.jar doc.ajf98.websim.WebSim
    inputfile.xml ../ws3.xsd
```

In the above commands, replace `wsthree.jar` with the location on your system of the WS^3 `.jar` file. Replace `inputfile.xml` with the location of the desired XML input file specifying the system you wish to simulate, and `inputschema.xsd` with the location of where the supplied XML schema file is (typically the same directory as the `.jar` file) — but with `../` prefixing it.

The Java virtual machine will initialize, and load and run WS^3. WS^3 will execute, outputting extra information according to the options you specified in the XML input file, and then terminate. It may output a number of files, as explained in greater detail in section A.10 on the facing page. If you want to abort the simulation whilst it is still executing, the key combination Ctrl-C should achieve this on most systems.

A.9.1 Data Dumping

WS^3 has the facility to dump data on the system status at certain user-specified intervals. These intervals are specified by the `dataDumpPeriod` attribute of the `<system>` element. This attribute specified how often data is dumped, in virtual seconds. The default is every 1 virtual seconds.

The data is dumped to a file which has a similar name and the same location as the input XML file, but has the string '`_dump.csv`' appended to the end. The name of this file is printed on the screen during the execution of WS^3.

A.9.2 Tracing

WS^3 has the facility to output a trace file during the execution of the simulation. This trace file describes actions that are occurring in the simulated system. It

[5]This problem is due to a limitation in the way `jar` files work — most Java virtual machines ignore the `CLASSPATH` environment variable when executing a `jar` file.

[6]You may have to adjust the syntax of the command slightly for your system.

can help you to understand how the simulation works if you are unsure, and can also be useful if you are not getting the results you expect.

The trace file is created if the `traceLevel` attribute of the `<system>` element is greater than 0 (see sub-section A.8.2 on page 78). The level of detail in the trace file is specified by that attribute. Experiment with the value of the attribute to get the level of detail you want.

The trace file is automatically created with a similar name and the same location as the input XML file, but has the string '`_trace.txt`' appended to the end. The name of this file is printed on the screen during the execution of WS^3.

A.10 Interpreting the Output of WS^3

WS^3 prints summary output on the screen once it is finished. This summary output should be self-explanatory to anyone familiar with simulation. It also prints tracing output as explained in sub-section A.9.2 on the facing page, which should be equally self-explanatory. Data is dumped to a CSV file (as explained in section A.9.1 on the preceding page, and this data can be imported into spreadsheets or data analysis tools. Each row represents a discrete point in virtual time, with the left-hand-most column indicating what virtual time that was. Each column represents a measurable parameter of a system object at that point in time. The first row of the CSV contains headings for these columns which should also be self-explanatory.

For more information on how the output of WS^3 works, see the full final report for this project [1].

Appendix B

Network Diagram Conventions

Figure B.1 gives a quick overview of the style of network diagrams used in this report. The diagram shows a client ($C1$) connected to a network node ($N1$). In fact, there are two connections between $C1$ and $N1$, as shown by the fact that the arrows is double headed. $N1$ is connected to a server called $S1$ which also has two connections — one from $N1$ to $S1$, and one running the other way.

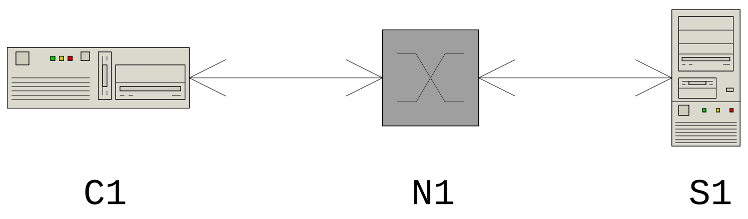

Figure B.1: Example Network Diagram

Appendix C

UML Diagrams

Figure C.1 on the next page gives a very quick overview, in one diagram, of the UML notation used in this report. Each box represents a class. Class `A` has four attributes — `p`, which is of type `P`, is public (it has a + sign), and has the value 3, `q`, which is of type `Q`, is private (it has a − sign), and has the value 4, `r`, which is of type `R`, is protected (implementation visibility only — it has a # sign), and has the value 5, and finally `s`, which is of type `S`, is public, is static (has class scope — it is underlined), and has the value 6. Class `A` also has two methods — `a`, which takes one parameter `u` of type `U`, and returns an instance of type `A`, and method `b` which takes no parameters and returns an instance of class `B`. Class `A` is abstract (the name of the class is italic).

Class `B` implements or extends `A`. Class `B` can have references to some instances of class `C`, and this is an aggregates relationship — the objects of class `C` are deemed to be part of the object of class `B`, although this relationship is not typically explicit in the programming language (it is not in Java).

Class `B` also has references to zero or more objects of class `E`, although this relationship is deemed to be less tightly coupled — again, this is not explicit in the language.

Class `D` directly inherits from class `B`.

Classes `D` and `E` are in package `F`.

For more information about UML, see [70, 71].

C.1 Conventions

UML diagrams in this report are drawn using the `dia` software [72]. The diagrams are not necessarily complete but only show portions of class hierarchies, and may be missing information to improve clarity.

The following conventions are used in the UML diagrams:

- In my UML diagrams, I do not distinguish between abstract class and interfaces. They both occur with the name of the class/interface in italics.

- Public class members (those prefixed with a +) are deemed to be visible to the whole program. Private class members (those prefixed with a −) are private to that class. Protected class members (those prefixed with a

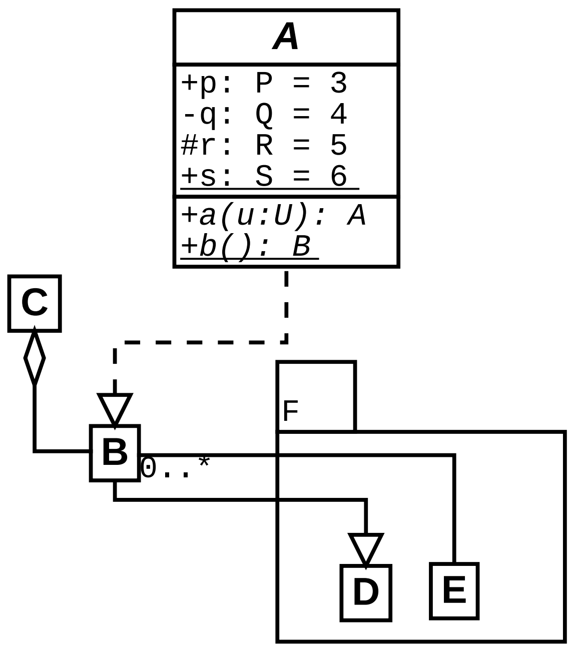

Figure C.1: Example UML Diagram

#) are visible only to that class and it's subclasses. Friendly (or package-private) class members (those without a specific prefix) are visible to that class and the package it is in (if it is in a sub-package, that sub-package only). For more information, see [73].

Appendix D

Bibliography

[1] Ferrier, Andrew. *MEng Individual Project — Web Server Performance Simulation* (2002). The website for this project and the associated software (WS^3). Contains downloadable reports, program binaries and source code, and other related items.
URL http://www.new-destiny.co.uk/andrew/project/

[2] Harrison, Peter. *Peter Harrison's Homepage, Department of Computing* (2002).
URL http://www.doc.ic.ac.uk/~pgh/

[3] Harder, Uli. *Uli Harder's Homepage, Department of Computing* (2002).
URL http://www.doc.ic.ac.uk/~uh/

[4] Field, Tony. *Tony Field's Homepage, Department of Computing* (2002).
URL http://www.doc.ic.ac.uk/~ajf/

[5] Sadri, Fariba. *Fariba Sadri's Homepage, Department of Computing* (2002).
URL http://www.doc.ic.ac.uk/~fs/

[6] Richardson, Tim. *UK census site stays shut* (2002). An article describing how the UK census site failed when it was first opened because of inadequate server sizing.
URL http://www.theregister.co.uk/content/6/23948.html

[7] Koller, Mike. *Ellis Isle: Uptime Comeuppance* (2001). A description of how the Ellis Island web site failed due to poor server sizing.
URL http://www.internetweek.com/newslead01/lead042601.htm

[8] *Definition of 'server farm'.*
URL http://www.pcwebopaedia.com/TERM/S/server_farm.html

[9] *WWW Consortium.*
URL http://www.w3.org/

[10] *Internet RFC/STD/FYI/BCP Archives.*
URL http://www.faqs.org/rfcs/

[11] Field, Tony. *Simulation and Modelling* (2000). Lecture notes from Simulation and Modelling course, lectured by Tony Field, 1999/2000 academic year.
URL `http://www.doc.ic.ac.uk/~ajf/Teaching/Simulation.html`

[12] Ferrier, Andrew, Morgan, Henry and Kay, Robert. *An Introduction to Queueing Theory* (1999). A useful tutorial and reference to queueing theory which I did with two Department of Computing colleagues in the first year.
URL `http://www.new-destiny.co.uk/andrew/past_work/queueing_theory/index.html`

[13] *Sun's Java Page*.
URL `http://java.sun.com/`

[14] W3C. *XML (eXtensible Markup Language)*.
URL `http://www.w3.org/XML/`

[15] W3C. *Who's Who at the World Wide Web Consortium*.
URL `http://www.w3.org/People/all#timbl`

[16] Menasce, Daniel E. and Almeida, Vergilio A.F. *Capacity Planning for Web Services*, pp. 23–63. Prentice Hall, Upper Saddle River, NJ 07458, 2002. ISBN 0130659037. A 'conventional' book on web capacity planning based on analysis and theory. Makes limited reference to web simulation.

[17] Nielsen, Jakob. *Response Time Overview* (1994). Excerpt from the book *Usability Engineering* by the same author.
URL `http://www.useit.com/papers/responsetime.html`

[18] Greenspun, Philip. *Philip and Alex's Guide to Web Publishing*, chap. 5: Learn to Program HTML (Hypertext Markup Language) in 21 Minutes. Morgan Kaufmann. ISBN 1558605347. Or available online.
URL `http://www.arsdigita.com/books/panda/html`

[19] *Macromedia Flash*.
URL `http://www.macromedia.com/software/flash/`

[20] Banks, Jerry, Carson II, John S., Nelson, Barry L. and Nicol, David M. *Discrete-Event System Simulation*. Prentice Hall. ISBN 0130887021. A good, detailed, up-to-date book.

[21] Kuhl, Frederick, Weatherly, Richard, Dahmann, Judith and Jones, Anita. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall. ISBN 0130225118.

[22] Law, Averill M. and Kelton, David W. *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 2000. ISBN 0070592926.

[23] Zeigler, Bernard P., Praehofer, Herbert and Kim, Tag Gon. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press. ISBN 0127784551.

[24] Jain, Raj. *The Art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley & Sons. ISBN 0471503363.

[25] Lewis, T.G. *Distribution Sampling for Computer Simulation.* Lexington Books, 1975. ISBN 0669971391.

[26] Mukherjee, Shubhendu S., Adve, Sarita V., Austin, Todd, Emer, Joel and Magnusson, Peter S. *Performance Simulation Tools.* Information on performance simulation of low-level hardware.

[27] *Simulation for Performance Measurements.*
URL `http://www.performance-simulation.com/`

[28] *Simulation Software Survey.*
URL `http://www.lionhrtpub.com/orms/surveys/Simulation/Simulation.html`

[29] Kennington, Alan. *Simulation Software Links.*
URL `http://www.topology.org/soft/sim.html`

[30] Murugesan, Sam and Deshpande, Yogesh (eds.). *Web Engineering: Managing Diversity and Complexity of Web Application Development.* Springer Verlag. ISBN 3540421300.

[31] Killelea, Patrick. *Web Performance Tuning: Speeding Up the Web.* O'Reilly. ISBN 1565923790.

[32] Hong, Duke P. *Constructing Web Server Performance Models* (1998).
URL `http://www1.ics.uci.edu/pub/duke/opnet98-web.pdf`

[33] *ACM Transactions on Modeling and Computer Simulation.*
URL `http://www.acm.org/tomacs/`

[34] Slothouber, Louis P. *A Model of Web Server Performance* (1995).
URL `http://www.geocities.com/webserverperformance/modelpaper.html`

[35] Masand, Brij and Spiliopoulou, Myra (eds.). *Web Usage Analysis and User Profiling.* WEBKDD'99 Workshop, Springer Verlag. ISBN 3540678182.

[36] Lu, Hongjun and Zhou, Aoying (eds.). *Web-Age Information Management Conference.* Springer Verlag, 2000. ISBN 3540676279.

[37] ACM. *ACM Digital Libraries.* ACM Press. ISBN 158113231X. See in particular the paper 'Server Selection on the World Wide Web'.

[38] Apache. *Apache Flood Tool.*
URL `http://httpd.apache.org/test/flood/`

[39] Mosberger, David and Jin, Tai. *httperf.* A tool for measuring web server performance by applying loads to a server.
URL `http://www.hpl.hp.com/personal/David_Mosberger/httperf.html`

[40] *Web Server Simulation* (1999). A Powerpoint Presentation on Web Server Models. Most of it is not relevant to this project, but they state 'Experimenting with actual server systems can be more expensive than simulation software' — the premise behind my project.
URL http://www.eos.ncsu.edu/eos/info/ie/ie441_info/arena/Semester%20Projects%20441/Fall_1999/Fall1999Presentations/Group1.ppt

[41] *Discussions with Peter Harrison.*

[42] *GCJ: The GNU Compiler for Java.* A compiler for Java that can compile to native code.
URL http://gcc.gnu.org/java/

[43] Apache. *Xerces 1.1.4 Software.* A JAXP-compliant Java XML Parsing toolkit.
URL http://xml.apache.org/xerces-j/index.html

[44] van der Vlist, Eric. *Using W3C XML Schema.*
URL http://www.xml.com/lpt/a/2000/11/29/schemas/part1.html

[45] *DTD Tutorial.* There appear to be no definitive web pages on DTDs but this page seems as good as any.
URL http://www.w3schools.com/dtd/default.asp

[46] Clifford, Peter. *Simulating Random Variables.* The page referred to here is for the inversion method, which I used for the Pareto distribution.
URL http://www.jesus.ox.ac.uk/~clifford/a5/chap1/node5.html

[47] Vázquez-Abad, Felisa J. and Champoux, Yanick. *Generation of Random Variables — The Pareto Distribution* (1998).
URL http://babyl.dyndns.org/SimSpiders/GenerRV/Distributions/pareto.html

[48] *Pareto Distribution.*
URL http://mathworld.wolfram.com/ParetoDistribution.html

[49] *The Pareto Distribution.*
URL http://www.math.uah.edu/stat/special/special12.html

[50] Rytgaard, Mette. *Estimation in the Pareto Distribution.*
URL http://www.casact.org/library/astin/vol20no2/201.pdf

[51] Rossi, Christian. *Various documents on IEEE standard 754 on Binary Floating-Point Arithmetic* (1999).
URL http://cch.loria.fr/documentation/IEEE754/

[52] Hollasch, Steve. *A simple explanation of IEEE standard 754 on Binary Floating-Point Arithmetic* (2001).
URL http://research.microsoft.com/~hollasch/cgindex/coding/ieeefloat.html

[53] Knuth, Donald E. *The Art of Computer Programming*, vol. 2: Seminumerical Algorithms, pp. 10–11. Addison-Wesley, 1997, 3rd edn. ISBN 0-201-89684-2.

[54] Jansson, Birger. *Random Number Generators.* Victor Pettersons Bokindustri Aktiebolag, 1966.

[55] Press, William H, Teukolsky, Saul A, Vetterling, William T. and Flannery, Brian P. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press. ISBN 0-521-43108-5.

[56] Coddington, Paul. *Monte Carlo Simulation for Statistical Physics.*
URL http://www.npac.syr.edu/users/paulc/lectures/montecarlo/

[57] *The Java Language Specification: Types, Values, and Variables.*
URL http://java.sun.com/docs/books/jls/first_edition/html/4.doc.html#9208

[58] *Debian GNU/Linux.*
URL http://www.debian.org/

[59] *SuSE Linux.*
URL http://www.suse.com

[60] *Javadoc Tool Home Page.*
URL http://java.sun.com/j2se/javadoc/

[61] *Code Conventions for the Java^{TM} Programming Language.*
URL http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

[62] *Fuzz.* A program for testing other programs by feeding them random input.
URL https://sourceforge.net/projects/fuzz

[63] Poskanzer, Jeffrey A. (2002). Software used.
URL http://www.acme.com/software/weblog_parse

[64] *Discussions with Uli Harder.*

[65] *Enterprise JavaBeans.*
URL http://java.sun.com/products/ejb/

[66] Stallman, Richard and McGrath, Roland. *GNU Make.*
URL http://www.gnu.org/software/make/make.html

[67] *Cygwin.* A Unix-like environment for Windows. It can be used to compile WS^3 on Windows using the supplied makefile.
URL http://www.cygwin.com/

[68] *Download Extensions* (2002).
URL http://java.sun.com/docs/books/tutorial/ext/basics/download.html

[69] *XML Spy.* An XML editing suite.
URL http://www.xmlspy.com/

[70] *UML* (2002). A fairly basic but still very useful guide and quick reference to UML notation.

[71] OMG. *UML Resource Page* (2002). Contains links to the UML specification and other UML-related information.
URL http://www.omg.org/uml/

[72] Larsson, Alexander and Henstridge, James et al. *Dia.* A drawing and diagramming program.
URL http://www.lysator.liu.se/~alla/dia/

[73] Ambler, Scott W. *Member function visiblity in Java programs* (September 2000).
URL http://www.ibm.com/developerworks/library/tip-mem-visibility.html

Appendix E

Glossary

Apache Refers to the Apache Sofware Foundation.

page 36

Erlang An Erlang is a unit of traffic intensity (which is represented with ϱ). It is named after A. K. Erlang, one of the creators of queueing theory.

page 44

event-based Event-based simulation is one of two main methods of computer-based simulation, which involves directing simulating events. See also process-based simulation.

page 14

farm A 'farm' or 'server farm' is a collection of interconnected serving machines, typically low-powered, which distribute the load of serving requests.

page 13

FORTRAN A programming language which has very strong mathematics capabilities.

page 36

gcj A native code compiler for Java which I used on Linux.

page 36

Kendall Kendall notation, such as M/M/1, is a notation used to describe queueing models. For more information, see [12].

page 44

Linux A free Operating System.

makefile A makefile makes it easier to build a program or parts of a program. WS^3 is supplied with a makefile which uses GNU make [66].

page 74

process-based Process-based simulation is one of two main methods of computer-based simulation, which involves simulating the processes which cause events. See also event-based simulation.

page 14

queueing network A queueing network is a network based around queueing: in it's most general sense, object or entities move around the network and wait in queues at places around the network. In WS^3, the objects are messages, and the queues are attached to servers and network nodes.

page 29

queueing theory Queueing theory involves using mathematical theories to model systems and determine various parameters of the system based upon the provision of other parameters of the system. It is an alternative to simulation of a system.

page 14

schema A 'schema' in general is a set of constraints, or a layout, for a set of data. In the context of the this project and WS^3, a schema normally refers to an XML Schema.

page 77

Xerces A JAXP-compliant XML parser for Java, written by Apache.

page 36

XML Schema A 'schema' which constrains the content of an XML file.

page 25

Appendix F

List of Acronyms

CSV	Comma Separated Variables A record-based file where each new field is delimited by a comma, and each record is delimited by a new line. They can be used for exchanging information between spreadsheets and similar programs.
DOM	Document Object Model An interface for accessing and updating documents, such as those written in XML.
DTD	Document Type Definition A file specifying the structure of an SGML file, of which XML is a subset. See [45] for more information.
EJB	Enterprise Java Beans A component architecture for Java [65].
FIFO	First In First Out A queueing discipline.
GNU	GNU's Not Unix A free software system.
GUI	Graphical User Interface A user interface using graphics and commonly 'windows', such as Microsoft Windows or XWindows.
HTML	Hypertext Markup Language The markup language used to create a typical web page.
HTTP	Hypertext Transfer Protocol The protocol used on top of TCP/IP to transfer web data — typically but not necessarily HTML pages.
JAXP	Java API for XML Parsing A method for accessing and updating XML documents in the Java programming language.
LAN	Local Area Network A network typically with no more than a few hundred hosts, distributed over a small (local) area, at most typically a building or two.
PC	Personal Computer A small computer designed primarily for personal use.

SGML	Standard Generalized Markup Language The more powerful and more complex precursor to XML.
TCP/IP	Transmission Control Protocol/Internet Protocol The general data transfer protocol used on internets and on the Internet.
TTL	Time-to-Live A TTL value is normally used in network to avoid problems caused by network routing loops: a TTL value is decremented each time a packet or message is routed — if it reaches zero, the packet or message is discarded. This involves over-flooding of the network. WS^3 emulates TTL values for all messages.
UML	Unified Modelling Language UML is a modelling language designed to model many aspects of software systems. It is based upon it's precursors, OMT, Booch, and OOSE. In this project UML has been used primarily to provide class hierarchy diagrams and other related items in the report.
WS^3	Web Server Simulation System The name of the software written for this project. The acronym WS^3 is sometimes written as WSSS.
WSSS	Web Server Simulation System See WS^3.
WWW	World Wide Web The interconnected hypertext web created by the worldwide network of HTTP servers, connected with TCP/IP.
XML	eXtensible Markup Language A generalised markup language. See [14] for more information.