A Mobile Agent Execution and Location Environment for the Perl Programming Language

A thesis presented

by

Paul Barry, IT Carlow

to

The Department of Accounting and Computing

in partial fulfillment of the requirements

for the degree of

Master of Science in Computing

Institute of Technology, Sligo

Ballinode, Sligo, Ireland

June 2003

©2003 – Paul Barry, IT Carlow

All rights reserved.

Thesis Supervisor

Dr. Jeanne Stynes, Cork IT

Author

Paul Barry, IT Carlow

Abstract

A Mobile Agent Execution and Location Environment for the Perl Programming Language, called *Scooby*, has been developed. Meeting the majority of its design goals, the delivered software provides for reliable, authenticated and securely encrypted transportation of mobile agents within a distributed computing environment. The Mobile Agent Paradigm offers a radically different model for software deployment within a distributed computing environment. Assuming the appropriate security mechanisms are in place, mobile agents can provide for truly dynamic Internet services. However, security concerns will likely hamper widespread adoption.

Contents

Title Page i
Abstract iii
Table of Contents iv
List of Figures vii
Acknowledgments viii
Dedication ix

1 Introduction **1**
1.1 Definitions 1
1.2 Aims and Objectives 2
1.3 Structure of this Document 3

2 Literature Review and Summary **5**
2.1 Understanding Mobile Agents 5
2.2 A Classification Scheme for Relocation Models 7
2.3 The Requirement for Mobile Agents 8
2.4 Mobile Agent Difficulties 10
2.5 Mobile Agent Applications 11
2.6 Summary of Existing Tools 12
2.6.1 D'Agents 12
2.6.2 Agents for Remote Access 12
2.6.3 Aglets Workbench 12
2.6.4 Ajanta 13
2.6.5 Agent Perl 14
2.7 A Critique of Agent Perl 14
2.8 Using Perl for Mobile Agents 16

3 Requirements and Specification **18**
3.1 Requirements 18
3.1.1 The Location 18
3.1.2 The Mobile Agent 19
3.1.3 Relocating Mobile Agents 19
3.1.4 Authentication 20
3.1.5 Safety 20
3.2 Functional Specification 20
3.2.1 Increments For The Location 20
3.2.2 Increments For The Mobile Agent 21

4 Design **25**
4.1 Introduction . . . 25
4.2 The Development Process . . . 25
4.3 The Relocation Mechanism . . . 27
4.3.1 Modify the Perl Interpreter . . . 27
4.3.2 Transmit the Perl Environment . . . 28
4.3.3 Transmit Perl Byte-code . . . 29
4.3.4 Modify the Mobile Agent Source Code . . . 30
4.4 The Security Mechanisms . . . 31
4.4.1 Protecting Mobile Agents In-Transit . . . 31
4.4.2 Malicious Mobile Agent Environments . . . 33
4.4.3 Malicious Mobile Agents . . . 33
4.5 The Transport Mechanism . . . 34
4.6 The Load-and-Execute Mechanism . . . 35

5 Implementing the Facility **36**
5.1 Introduction . . . 36
5.2 Solving the Central Problem . . . 37
5.2.1 A Matter of Trust . . . 40
5.2.2 A Mutating Automatic Debugger . . . 41
5.2.3 What Happens: Step-by-Step . . . 42
5.2.4 The Web-based Monitoring Services . . . 45
5.3 Testing and Debugging . . . 45
5.3.1 The Test Network . . . 47
5.3.2 The Ordering of Events within a Distributed System . . . 47
5.3.3 Sample Network Traces . . . 52

6 Analysis **70**
6.1 Satisfying the Overall Goal . . . 70
6.2 Satisfying the Requirements . . . 71
6.3 Linux Only Implementation (For Now) . . . 75
6.4 Satisfying the Design . . . 75
6.4.1 The Delivered Relocation Mechanism . . . 75
6.4.2 The Delivered Security Mechanism . . . 76
6.4.3 The Delivered Transport and Load-and-Execute Mechanisms . . . 76
6.5 Critical Assessment . . . 77
6.5.1 How Strong is Scooby? . . . 77
6.5.2 How Strong is the Key Server? . . . 77
6.5.3 What Actually Happens: An Example . . . 78
6.6 Future Enhancements . . . 81
6.6.1 Enhancing the Key Server . . . 81
6.6.2 Enhancing `Scooby.pm` . . . 82
6.6.3 Enhancing `Location.pm` . . . 82
6.6.4 Enhancing `Executive.pm` . . . 83

7 Conclusions **84**
7.1 Some Personal Comments . . . 84
7.2 Unanswered Questions . . . 85
7.2.1 Are Mobile Agents Useful? . . . 86
7.2.2 Will Mobile Agents be Pervasive? . . . 86

Bibliography **88**

List of Figures

2.1 A Classification Scheme for Relocation Models 8

5.1 The Test Network at The Institute of Technology, Carlow. 48

Acknowledgments

The Perl programming community is all about sharing code, ideas, experiences and technologies. The community openly encourages Perl programmers to build upon the software components available from the Comprehensive Perl Archive Network (CPAN). I would like to thank the following Perl programmers for their contributions to the CPAN archive: Vipul Ved Prakash (`Crypt::RSA`), Hiroyuki Oyama (`Net::MySQL`), Raphael Manfredi (`Storable`), Gisle Aas (`libwww-perl`) and Robin Houston (`PadWalker`). These modules provide essential services to the software delivered as part of this project. If it was not for their general availability, I would still be coding with no real end in sight.

My standard writing environment includes such tools as `vi` and LaTeX, running on Linux. So, thanks to Bill Joy, Leslie Lamport and Linux Torvalds, respectively, for giving the world these great tools. Thanks are also due to Larry Wall, the original author of Perl, and the Perl 5 Porters, who now maintain the language and its environment. The more I work with it, the more I find Perl to be a fantastic, as well as fascinating, programming technology.

Thanks to Alex Barnett for providing a freely downloadable thesis class file for LaTeX, called `huthesis`. Alex's format is used by the Harvard University Physics Department and is based on the University of California Ph.D. dissertation format (`UCTHESIS`). I have made only minor changes to the `huthesis.cls` disk-file, which I used in the production of this document.

Thanks to Dr. Jeanne Stynes, Cork Institute of Technology, who (at very short notice) agreed to supervise this project. I am grateful for Dr. Stynes' patience and advice, as well as her (perfectly timed) "when will you finish?" e-mails. Thanks are also due to Dr. Michael Madden, NUI Galway, for agreeing to act as external examiner to this project.

Finally, a big thank-you to my wife, Deirdre, who has had to endure yet another large, extra-curricular, time-consuming project.

Dedicated to all the great
and generous Perl programmers
who contribute to the CPAN repository

Chapter 1

Introduction

Students of the taught M.Sc. Science in Computing Programme, which is run under the auspices of the Higher Education Staff Development Network, are required to undertake a project/dissertation. This document discusses an area of study, **Mobile Agents**, and describes the delivered project, **A Mobile Agent Execution and Location Environment for the Perl Programming Language**. Upon completion of the taught modules of the programme, students are required to produce a thesis that draws on the taught material[1]:

> *"It is expected that the knowledge, skills and techniques gained in the taught modules can now be applied and integrated into your project/dissertation and be complementary to your current duties and responsibilities."*

The taught modules successfully completed are:

1. Object Oriented Systems Development,
2. Software Engineering & Project Management,
3. World Wide Web Design & Development, and
4. Computer Networks.

1.1 Definitions

A *mobile agent* is a *network-aware* software application that can, under certain circumstances, *suspend* its execution on one network device, *transport* itself to another network device and then *resume* execution once there.

[1] From the *General Guidelines* to candidates on the Training of Trainers Masters Degree Programmes.

A mobile agent has two main components: *code* and *state*. For a software application to relocate from one network device to another, its code has to somehow *travel*. The process of traveling is referred to as *relocation*. Typically, "the code" that relocates is in source code form, compiled form or in some intermediate, byte-code form. The *state* also has to travel. Here, "state" refers to the resources the software application is currently using, which includes (but is not limited to), memory variables, open disk files, network sockets, printer connections and (in some cases) registers. The entire set of resources is referred to as the software application's *state*.

A *mobile agent environment (MAE)* is a set of technologies that provide a working space for mobile agents. A MAE has to exist on each network device that the mobile agent can execute on (and relocate to), and this can include the device that starts the mobile agent. MAEs are often referred to by a number of names, with *site*, *location* and *place* the most popular monikers. Throughout this document, "Location" is used to refer to the MAE.

1.2 Aims and Objectives

The overall aim of this project is to extend the Perl programming language to allow for the development of portable, operating system-independent, mobile agents. Specific objectives are the development of a number of add-on modules to Perl, including:

- `Mobile::Executive` - a module that allows for the development of mobile agent programs.
- `Mobile::Location` - a module that allows for the development of Locations to which mobile agents can be transported, and re-executed.

The modules developed will support a *hybrid API* - support will be provided for both procedural and object-oriented programmers. It is proposed to develop these Perl software extensions using the *Incremental Model*, as detailed in [Pressman, 1997]. The Incremental Model combines the *Linear Software Development Model* with *Prototyping*.

1.3 Structure of this Document

This document describes a mobile agent execution and location environment for the Perl programming language, called *Scooby* (also referred to as "the facility" throughout the remainder of this document). Separate chapters detail the completed literature review (which includes a summary of some existing tools), the facility's requirements specification, and design. The delivered facility is discussed at some length, with detailed explanations of its inner workings given in the chapter that discusses the implementation of the facility. Testing the facility proved to be troublesome, and the implementation chapter also discusses the typical type of problem encountered. Additional chapters provide an analysis of the delivered facility, as well as a series of conclusions. An bibliography concludes this document.

A detailed description of the Perl programming language can be found in [Wall, 2000]. The most recent version of the facility's source code, documentation and installation instructions is available from the *Scooby* website, located at:

`http://glasnost.itcarlow.ie/~scooby/`

The accompanying CD contains the following material:

Installation Instructions - contained in `install.pdf`, this document provides step-by-step installation instructions.

Annotated Source Code - contained in `annotated.pdf`, this document presents a detailed, line-by-line explanation of the delivered source code.

Source Code Archives - `Mobile-Location-4.02.tar.gz`, `Devel-Scooby-4.12.tar.gz`, `keyserver-1.04.tar.gz` and `Mobile-Executive-2.03.tar.gz` contain zipped/tarred archives of each of the delivered software components.

The Scooby Guide - contained in `TheScoobyGuide.pdf`, this document provides a programming guide for writers of Mobile Agents and Locations.

The On-line Documentation - contained in `onlinedoc.pdf`, this document contains a typeset, printable version of the on-line documentation (which is available as UNIX man pages after installation).

Examples - contained in `examples.pdf`, this document contains the source code to a selection of mobile agents developed during the lifetime of this project.

Thesis - contained in `thesis.pdf`, this is an electronic copy of this document. Note that this disk-file is designed to be printed, *not viewed on screen.* When viewed on screen, the font may not mimic that of the printed page.

Chapter 2

Literature Review and Summary

This chapter presents a review of the literature surrounding the study of mobile agents, together with a summary of some existing mobile agent programming technologies. A critique of another Perl-based mobile agent technology, *Agent Perl*, is also offered.

2.1 Understanding Mobile Agents

[Cockayne, 1998] states:

> *"A mobile agent is a program with the ability to move during execution, while preserving its identify and state ... [it is] a solution that satisfies the basic needs of increased sophistication without a crippling sacrifice in bandwidth."*

A Mobile Software Agent[1] is a specific type of *Software Agent*. A number of definitions exist for Software Agent (see [Franklin, 1996]), with each author (or research group) seeming to favour their own interpretation of what is meant by Software Agent Technology. Certain characteristics commonly appear, and a good review of these characteristics can be found in [Sundsted, 1998].

Mobility is the main characteristic of a mobile agent, and the study of *code mobility* as it relates to the Perl programming language forms the basis of this project. For a software program to exhibit the mobility characteristic, it must be capable of deciding - *or being told* - to move to some other computing machine to continue its work.

[1] More generally referred to as *Mobile Agent*.

To illustrate the complexities involved in realising this conceptually simple idea, let us assume that a mobile agent is currently executing on computer A, and that it is attempting to move to computer B. When an instruction to relocate occurs, the mobile agent is required to perform a sequence of steps similar to the following:

- **Determine if it possible to move from A to B**: is B operational and is B willing to accept the mobile agent at its location?
- **Save the current execution state as it exists on A**: if the mobile agent has created variables or data structures on A, it is necessary to save their *state* prior to transportation. Additionally, the current state of the execution stack may need to be saved if the request to transport occurs within a subroutine. If files are open on A, they may need to be closed and (if appropriate) transported to B.
- **Relocate from A to B**: the executable code, together with the execution state, needs to be transported to B in a reliable fashion.
- **Recreate the execution state on B**: the transported variables and data structures need to be created and set to the values they had on A. The execution stack may need to be reinstated. If files accompany the agent, they must be reopened on B. Finally, B needs to load the executable code, and begin executing at the point in the code *immediately following* the request to transport.

It should be clear that the method of operation as detailed above is not typical of today's software programs or programming environments. A thorough description of the components required within the Mobile Agent Paradigm is presented in Chapter 2 of [Cockayne, 1998].

The capturing of state is well understood at the level of the operating system - the technology required to swap executing programs into and out of virtual memory is well understood, as is the technique of capturing the state of a running program so that interrupts can be promptly handled. It is quite another thing to have a currently executing program capture its own state, transport itself to another computer (which may be running

a different operating system) and continue executing from where it left off. This relocation process is often referred to as *code migration.*

2.2 A Classification Scheme for Relocation Models

The classic paper on code migration as it relates to mobile agents is [Fuggetta, 1998]. Two characteristics are identified as important: the *mobility* characteristic and the *initiator* characteristic.

The *mobility characteristic* refers to that which actually relocates. With **weak** mobility, only the mobile agent code (and, possibly, some initialisation data) is transported. Weak mobile agent systems provide a means to relocate a program, then restart the same program from its initial state. Conversely, **strong** mobility refers to the transportation of code *and* state, providing a means whereby a relocated program can continue to execute on the relocated-to machine, effectively continuing from where it left off on the previous machine.

The *initiator characteristic* refers to which end of a network connection requests the relocation. If the receiving-side of a networked connection requests relocation, it is said to provide for *receiver-initiated relocation.* The target (or server) machine sends a program to another computer for execution. The Java applet facility falls into this category. With *sender-initiated relocation*, the machine currently executing a program decides to relocate the program to another computer. The program is sent to the target (or server). D'Agents (to be discussed shortly) falls into this category.

Figure 2.1, on page 8, summarises the various alternative relocation mechanisms[2]. It is possible to have a sender or receiver-initiated weak mobility mechanism, just as it is possible to have a sender or receiver-initiated strong mobility mechanism. [Tanenbaum, 2001] further describes how it is possible to further classify the behaviour of the relocation mechanism based on whether or not the relocated program executes within an existing process or a new process (when weak) or within a migrated or cloned process (when strong).

[2]Figure 2.1 is taken from page 162 of [Tanenbaum, 2001].

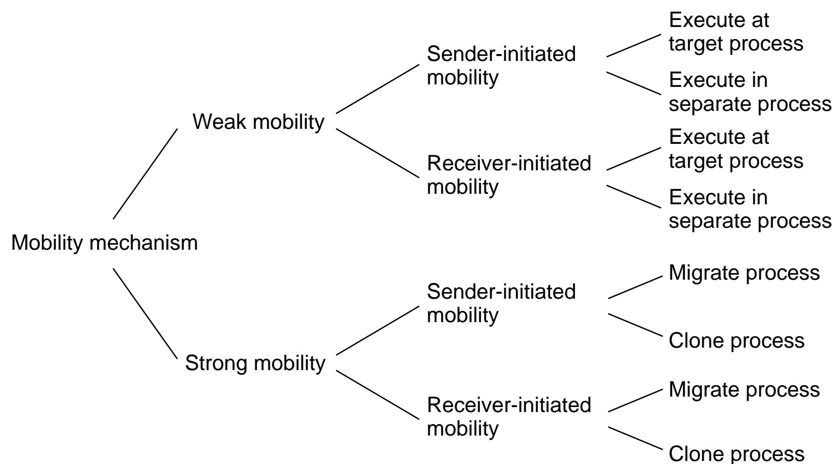

Figure 2.1: A Classification Scheme for Relocation Models

It is generally believed that developing a system that supports a sender-initiated, strong relocation mechanism is the most challenging. It is a goal of this project to implement just such as mechanism for the Perl programming language.

2.3 The Requirement for Mobile Agents

[Lange, 1999] identifies seven good reasons for using and deploying mobile agents within a distributed computing environment:

1. **Reducing network load** - traditional client/server systems often generate considerable network traffic, especially those that process vast quantities of remote data. Mobile agents allow the processing to occur where the data resides, effectively saving the bandwidth required to transport the data from its source machine to another prior to processing.

2. **Overcoming network latency** - controlling certain “real-time” equipment over a traditional network can result in latencies that may be intolerable. By relocating to “beside” or “within” the equipment to be controlled, latency problems can be avoided.

3. **Encapsulating protocols** - changing established protocols on the Internet is not easy[3]. Modern client/server systems implement protocols rigidly, and any updates to the protocols require changes to all clients and servers in the system, resulting in "legacy protocol problems". As mobile agents are adaptable, they can, under certain circumstances, overcome these problems.

4. **Executing asynchronously and autonomously** - being able to disconnect from its initial host and roam within a network environment removes the need for the application (encapsulated within a mobile agent) to maintain (often expensive) network connections for extensive periods of time.

5. **Adapting dynamically** - while roaming, mobile agents can directly interact with their environment, reacting and adapting dynamically to their surroundings.

6. **Being naturally heterogeneous** - as most mobile agents are computer and transport-layer independent (due mainly to their reliance on interpreted environments), they can offer a platform that is seamless and that can more readily support systems integration.

7. **Improving robustness** - the ability to react to their environments can allow mobile agents to offer a more fault tolerant service to applications. Consider a computer that is shutting down and warns all applications that they have three minutes to terminate. On a traditional system, this results in the services offered by the applications going off-line. On the other hand, a mobile agent has enough time to react to the terminate request by relocating to another Location and continuing to execute there.

IBM, an organization that continues to invest heavily in agent technology (most notably in its continued development of the *Aglets Workbench*, discussed below), believes in mobile agents, as described in [Chess, 1995]. Although the authors of this paper conclude that

[3]Consider, for example, the Simple Mail Transport Protocol (SMTP) and the RFC 822 7-bit ASCII message format still employed by all modern e-mail systems. Although an upgrade to RFC 822 is possible, the sheer size of the Internet makes such a global undertaking next to impossible, as *every* Internet-based e-mail server would need to upgrade *at the same time*.

everything that can be done with mobile agents can be achieved with traditional techniques, the IBM researchers demonstrated that the vast majority of mobile agent solutions bested the traditional solutions in practically all cases.

The Knowbot Research, described in [Hylton, 1996] and [Hylton, 1997], and conducted at the Corporation for National Research Initiatives in the USA, supports the claims made by IBM[4].

2.4 Mobile Agent Difficulties

The authors of the mobile agent position paper from the D'Agents Project (discussed below), are keen proponents of the mobile agent paradigm, as described in [Kotz, 1999]. However, they are not zealots. They identify two main categories of difficulties (or hurdles) that need to be overcome in order to allow for the widespread adoption of mobile agent technologies. The two categories are technical and non-technical. The technical difficulties include:

1. **Performance and scalability** - the mobile agent model turns the client/server model on its head. Processing that used to occur on the client is now shifted to the server. This can lead to scalability issues on the server. Additionally, the *migration overhead* involved in relocation can sometimes result in mobile agent solutions performing slower than their client/server counterparts.

2. **Portability and standardization** - the abundance of mobile agent systems, all based on different programming languages and interpreters, are hampering interoperability efforts. It is not possible, for example, to execute a mobile agent written for D'Agents within the Aglets Workbench[5] .

3. **Security** - the fact that a mobile agent is "foreign code" freely executing within a Location introduces a series of security problems. How does a Location protect against

[4] Refer to `http://www.cnri.reston.va.us/home/koe` for more details on Knowbot.

[5] Some efforts are underway to address this issue. For example, the SeMoA Project (see `http://www.semoa.org`) provides a mechanism to execute agents based on other technologies within their server. This includes the ability to execute mobile agents based on Aglets and JADE (`http://sharon.cselt.it/projects/jade/`).

a malicious mobile agent? How does a mobile agent protect itself against a malicious Location? These are difficult questions to answer, and considerable research into the answers is ongoing.

The non-technical difficulties include:

1. **Lack of a killer application** - there is no mobile agent application that cannot be developed the client/server way. And there is certainly no *killer application* that makes the choice of the mobile agent paradigm a non-issue.

2. **Getting ahead of the evolutionary path** - getting Internet service providers and sites to move en-masse to the mobile agent mode of operation is unlikely to occur. Providing an evolutionary path that allows ISPs and sites to gradually offer mobile agent based services is seen as a priority.

3. **Revenue and image** - as many Internet sites generate revenue from advertisements and the number of viewings of those advertisements (by a human user), it is unlikely that they will adopt a technology that allows their sites to be visited by an automated agent. Different revenue models need to be developed to allow mobile agents into such sites, while maintaining a revenue stream for the site owners.

2.5 Mobile Agent Applications

Most application that exists within the client/server world can be developed as a mobile agent. [Lange, 1999] identifies a collection of application types that suit the mobile agent model. These application types include: e-commerce, personal assistants, secure brokering, distributed information retrieval, telecommunications network services, work-flow applications, groupware, monitoring, information dissemination and parallel processing. Other examples of the use of the mobile agent paradigm include the use of the technology to assist in systems administration and network management, as described in Chapter 6 of [Barry, 2002].

2.6 Summary of Existing Tools

A number of mobile agent development technologies exist. A brief survey follows.

2.6.1 D'Agents

Developed at Dartmouth College.

Website: `http://agent.cs.dartmouth.edu`

Based around the Tool Command Language (TCL), D'Agents (previously known as Agent TCL) uses a modified version of the TCL interpreter to provide an environment within which mobile agents can operate. Migration is achieved by the `agent_jump` command, and the TCL interpreter has been modified *at the source code level* to allow `agent_jump` to manage the bundling and transportation of execution state and code. Within the mobile agent community, D'Agents is the best known and most widely cited mobile agent technology.

2.6.2 Agents for Remote Access

Developed at The University of Kaiserslautern.

Website: `http://wwwagss.informakik.uni-kl.de/Projekte/Ara/index_e.html`

The goal of Agents for Remote Access (ARA) is to provide an operating environment within which to execute mobile agents. A number of APIs have been developed to allow programmers to interact with ARA and move freely and easily within the ARA system, and from machine to machine. ARA-compliant mobile agents can be written in a number of programming languages, which include TCL, C/C++, and Java. In many ways, ARA can be considered to be a mobile agent operating system.

2.6.3 Aglets Workbench

Developed at IBM's Tokyo Research Laboratory.

Website: `http://www.trl.ibm.co.jp/aglets/`

Built around the Java programming language, IBM's Aglets Workbench provides a visual development environment for creating mobile agents that search for, access, and manage

data within network-centric environments. Java - a programming language that was built from the ground-up to support code mobility - has attracted considerable attention from the mobile agent community. See, for example, [Lange, 1998].

2.6.4 Ajanta

Developed at the University of Minnesota.

Website: `http://www.sc.umn.edu/Ajanta/`

Like Aglets, the Ajanta system is based on the Java programming language. What makes Ajanta noteworthy is the set of security mechanisms that it provides. The Ajanta system can detect a "damaged" mobile agent. The three Ajanta mechanisms that can be used for damage detection are:

1. **Examining Read-Only State** - the initiating network device digitally signs the state of the mobile agent (with a *private key*) in such a way that the receiving Location can determine if the state has been tampered with. If it has, the mobile agent is considered damaged.

2. **Employing Append-Only Logs** - as a mobile agent travels from Location to Location, information can be added to the state but it *cannot* be removed or modified without the final destination Location being able to detect the removal or modification (and then take any appropriate action). Again, digital signatures are employed by this mechanism.

3. **Supporting Selective Revealing** - encryption technologies allow for a data structure (typically an array) to have each of its components digitally signed by the *public-key* of each of the Locations visited. The entire data structure is also digitally signed by the initiating Location. Such a mechanism allows each of the *trusted* Locations to change that part of the data structure that has been encrypted with their *public-key* but nothing else. In this way, any of the Locations can easily determine if the mobile agent's state has been damaged and take whatever appropriate action is necessary.

2.6.5 Agent Perl

Developed at Carleton University.

Website: `http://epn.nu/~spurkis/Agent/`

While studying as an undergraduate, Steve Purkis developed a mobile agent extension to Perl called Agent Perl. Although an unfinished, proof-of-concept technology, Agent Perl nonetheless provides a mechanism for Perl programmers to create *transportable mobile agent objects* which encapsulate code and state information.

2.7 A Critique of Agent Perl

As long as the Perl programmer uses objects exclusively, Agent Perl can form the basis of a mobile agent system. Procedural programmers, however, receive no support from Agent Perl. Programmers from other disciplines (most notably Java) do not find such an object-centric approach restrictive. However, *it is* a major limitation for many Perl programmers. Perl encourages programmers to develop their code in whichever method is best for the programmer. Perl does not force the programmer to choose between the object-oriented approach and the procedural approach - Perl supports both. Agent Perl gives the programmer no choice - it's a case of use objects, or do nothing. The lack of a procedural interface to Agent Perl is a serious shortcoming.

Purkis' choice of a purely object-oriented approach is easy to comprehend - in Perl, *objects encapsulate their entire state* and packaging everything as an object is a tidy abstraction. By employing a mechanism that attaches Perl code to an object, it is then a trivial exercise to transport the object (data and code) to another machine and re-execute it, thus satisfying the definition of a sender-initiated mobile agent. However, Agent Perl does not support strong relocation. Re-execution is triggered at a specific entry point to the object, which positions Agent Perl mid-way between a weak and strong relocation technology.

Another shortcoming is the mechanism by which Agent Perl is used - the API

is complex, and requires the programmer to write far more code than is necessary[6]. To illustrate, consider the following sample program taken from the 3.20 release of the Agent Perl distribution. This program transports itself to a remote computer and prints "Hello World" on the screen of that computer, assuming the remote computer is running a *Static Agent* (that is, an Agent Perl Location):

```
#!/usr/bin/perl

package Agent::HelloWorld;

@ISA = qw( Agent );

sub new {

    my ($class, %args) = @_;
    my $self = {};

    foreach (keys(%args))
    {
        $self->{"$_"} = $args{"$_"};
    }

    bless $self, $class;
}

sub agent_main {

    my ($self, @args) = @_;

    my $to = delete($self->{Host});

    print "Are we there yet? " if $self->{verbose};

    unless ($to) {
        print "Hello World!\n";
        return 1;
    }

    print "no.\n Dispatching HelloWorld agent... " if $self->{verbose};

    my $msg = new Agent::Message(
                                     Body      =>
                                         [ "Run me\n", $self->store() ],
                                     Transport => TCP,
                                     Address   => $to
                                );

    if ($msg->send)
    {
        print "done.\n";
    }
```

[6] Something that is referred to as "not very Perlish".

```
        else
        {
            print "couldn't send agent!\n";
        }
    }
    1;
```

Most Perl programmers, even those well-versed in Perl's object-oriented techniques, need to spend some time reading the code presented above before it becomes understandable. It is incomprehensible to Perl programmers that program using the procedural approach.

The Perl interpreter cannot run the above code as it is - another program has to be created which reads the above code and executes it. Just such a program, called `ex.pl`, is provided with the Agent Perl distribution. To execute the `HelloWorld` agent and ask it to transport to the Location executing at port `20000` on the machine at IP address `149.153.100.67`, the following command is issued at the operating system command prompt:

```
perl  ex.pl  -n  HelloWorld  149.153.100.67:20000
```

This command launches the mobile agent, ultimately resulting in the message appearing on the screen of the indicated Location. Additional examples of Agent Perl in action are presented in Chapter 6 of [Barry, 2002].

2.8 Using Perl for Mobile Agents

Nigel Chapman, a leading Perl author, has this to say of Perl:

> *"Sometimes, it may seem as if Perl deliberately trails its coat in front of the mainstream programming languages, trying to start a fight."* [Chapman, 1997]

Perl is a controversial programming language. Perl's motto - *There's more than one way to do it!* - sums up the languages' greatest strength (in the eyes of its many fans), and its' greatest weakness (in the eyes of its numerous detractors). Rather than placing restrictions on programmers, Perl allows solutions to algorithms to be expressed in numerous ways. There is no single, correct way to do things in Perl - whatever works for the programmer,

works for Perl. Experienced programmers can find this approach liberating, whereas novice programmers often find the multitude of choices confusing.

As a combination of `ack`, `sed`, C, and shell programming, Perl can do many things. Perl has always been popular with systems administrators, and use of Perl exploded with the popularity of the World-Wide-Web, and in particular, the use of Perl as a programming language for the Common Gateway Interface (see [Meltzer, 2001] for a good overview). Some of the largest sites on the Internet - including Yahoo, Amazon and Google - make extensive use of Perl when developing dynamic web content and performing systems administration, as described in [O'Reilly, 1999].

However, Perl is not just a programming language for the web - it is a full-featured, general-purpose programming language, with excellent support for Internet protocols and programming. And Perl has a huge following. What is especially interesting is that Perl has achieved its success without the assistance of a multi-million dollar advertising campaign. No single commercial organization champions Perl - it is the result of an individual's need for a new tool, the classic "scratching an itch" popularised by the open source community.

With such a large community of Perl programmers developing software on the Internet, it makes sense to try and empower these developers to use Perl to create mobile agents, as attempts to convince long-time Perl programmers to switch to Java or TCL for mobile agent development are unlikely to be fruitful.

Chapter 3

Requirements and Specification

This chapter presents the requirements and functional specification for the facility.

3.1 Requirements

The requirements are broken down into five main categories:

1. The Location Requirements,
2. The Mobile Agent Requirements,
3. The Relocating Requirements,
4. The Authentication Requirements, and
5. The Safety Requirements.

Each of these categories is briefly expanded upon within the sections that follow.

3.1.1 The Location

It might seem obvious, but before a mobile agent can relocate from one host on a network to another, the mobile agent needs somewhere to go. The *Location* provides a mobile agent portal on a networked host. The job of the Location is to passively wait (forever) for mobile agents to arrive at its portal. When a mobile agent does arrive, the Location will perform a number of actions, including, but not limited to, the following:

1. Authenticate the mobile agent,

2. Accept the mobile agent (i.e., receive it from the network), and
3. Execute the mobile agent safely.

It is useful to think of the Location as conforming to the traditional role of the *server* within the client/server paradigm, as described in [Comer, 1999]. The main difference is that the Location may not provide any particular service to clients (other than the basic service which executes an authenticated and accepted mobile agent safely). This is not to say that other services cannot be provided. It should be possible to embed the Location within another network service (or server) and then provide a mechanism for mobile agents to interact with the service provided by the embedding application. Example embedding applications might include a web or e-mail server.

3.1.2 The Mobile Agent

If the Location conforms to the role of *server*, then the mobile agent conforms to the traditional role of the *client*. The mobile agent initiates communication from its current host (which may or may not be a Location) to some Location that it wishes to relocate to. The mobile agent authenticates were appropriate, and then transports itself to the Location. It then ceases to execute on its current host and continues executing (from where it left off) on the new Location.

3.1.3 Relocating Mobile Agents

The mobile agent can relocate from one host to another, or from one Location to another. To work well, this relocation must be reliable. Within an internetwork, this reliability can be achieved using the Transmission Control Protocol as the underlying transport mechanism. However, higher level protocols may also provide a relocation mechanism, for example, the Hyper-Text Transfer Protocol may be used to relocate to web servers, or the Simple Mail Transfer Protocol may be used to relocate to e-mail servers.

3.1.4 Authentication

We must be able to ensure that the Location relocated to is in fact the Location to relocate to. There is also a need to be sure that the mobile agent being received at a Location is arriving from a trusted Location. Various mechanisms already exist to help with this, and could include employing digital signatures based on PGP, which is described in [Garfinkel, 1995] and [Zimmermann, 1996].

3.1.5 Safety

The ability of mobile agents to relocate introduces a number of security issues, which can be categorized into two main areas of concern:

1. Protecting a Location from a malicious mobile agent, and
2. Protecting a mobile agent from a malicious Location.

Finding solutions which adequately address these two concerns is an area that is attracting considerable ongoing research within the mobile agent developer community.

3.2 Functional Specification

In the sections which follow, the functionality of the delivered system is described through a number of Location and mobile agent increments.

3.2.1 Increments For The Location

As discussed above, the Location provides a *mobile agent portal* on the network. A number of increasingly sophisticated increments of the Location technology will be delivered, and these are briefly described in the subsections which follow.

Increment #1: An Unsafe Location

Accept the mobile agent at an agreed protocol port, then execute the mobile agent. No authentication is performed, and the mobile agent is executed in a non-safe environment (i.e., no checks are performed on the intentions of the mobile agent).

Increment #2: A Safe Location

Execute the accepted mobile agent in a "safe environment". The standard distribution of Perl includes a module called `Safe.pm` which provides a mechanism to compartmentalize code in a sand-box prior to execution. The previous increment will be extended to provide support for `Safe.pm`.

Increment #3: An Authenticating Location

Only execute accepted mobile agents that have been successfully authenticated to the Location. The selected authentication mechanism (PGP digital signatures or some other suitable technology) will be integrated into the previous increment to support authentication of both mobile agents and Locations.

Increment #4: An Embedded Location

Providing access to services within Embedding Applications. The open-source web server, Apache[1], will be extended to support an Embedded Location. The Embedded Location will be based on the previous increment.

3.2.2 Increments For The Mobile Agent

For a mobile agent technology to support the Perl programming language, we need to be able to relocate simple Perl statements, the various Perl variable types, data structures, files, Perl objects, and (under certain circumstances) Perl modules. A series of mobile agent increments to accomplish this are described below.

Increment #1: A Simple Perl Mobile Agent

Relocate simple Perl statements (i.e., no variables, data structures or files), for example:

```
print ‘‘Hello on the original Location\n’’;
relocate( ‘elmo.itcarlow.ie’, 20000 );
print ‘‘Hello on the remote Location\n’’;
```

[1]See `http://www.apache.org` for more details.

will print the first message to standard output on the initiating host, transport the mobile agent to port 20000 on the machine called “elmo.itcarlow.ie” and then print the second message to standard output.

Increment #2: Relocating Variables

The various types of Perl variables will be dealt with systematically by the following increments.

Relocating Scalars

Relocate scalar variables, for example:

```
$message = ‘‘Did this print?\n’’;
relocate( ‘elmo.itcarlow.ie’, 20000 );
print $message;
```

will transport the mobile agent to port 20000 on “elmo.itcarlow.ie” then print the contents of the scalar variable `$message` to standard output.

Relocating Arrays/Lists

Relocate array/list variables, for example:

```
@countdown = ( ‘5’, ‘4’, ‘3’, ‘2’, ‘1’, ‘lift-off!’ );
relocate( ‘elmo.itcarlow.ie’, 20000 );
foreach $count (@countdown)
{
    print $count, ‘‘\n’’;
}
```

will transport the mobile agent to port 20000 on “elmo.itcarlow.ie” then print the countdown to standard output (over six lines).

Relocating Associative Arrays

Relocate associated arrays[2], for example:

[2]Referred to as a *hash* in the Perl community.

```
%pairings = (
              'tom' => 'jerry',
              'woody' => 'buzz',
              'kermit' => 'miss piggy',
              'barney' => 'bj'
            );
relocate( 'elmo.itcarlow.ie', 20000 );
while (($left, $right) = each ( %pairings ))
{
   print ''$left is associated with $right\n'';
}
```

will transport the mobile agent to port 20000 on "elmo.itcarlow.ie" then print the values of the pairings to standard output (over four lines).

Relocating References

Relocate referenced variables, for example:

```
@countdown = ( '5', '4', '3', '2', '1', 'lift-off!' );
$countref = \@countdown;
relocate( 'elmo.itcarlow.ie', 20000 );
print @$countref[2], ''\n'';
```

will transport the mobile agent to port 20000 on "elmo.itcarlow.ie" then print the value at array index 2 of the `@countdown` list to standard output - the value printed will be '3', as Perl starts counting array indices from zero.

Increment #3: Relocating Perl Objects

An *object* in Perl is simply a *reference* that happens to know the type of object it is pointing to, i.e., its *class*. Typically, in Perl, objects employ an associative array as the internal data structure. It should be possible to define the mechanism for relocating Perl objects based on the previous increments.

Increment #4: Relocating Open Files

A file which is in a opened state on the initiating host needs to be relocated to the Location along with the mobile agent. For example:

```
open( LOGFILE, '>log.txt' )
    or die ''Log file could not be opened: $!\n'';
print LOGFILE ''We are on the initiating host\n'';
relocate( 'elmo.itcarlow.ie', 20000 );
print LOGFILE ''We are now on the Location\n'';
close( LOGFILE );
```

will result in the file named "log.txt" residing on the Location (which is "elmo.itcarlow.ie"). The file will have two lines as its contents, as follows:

```
We are now on the initiating host
We are now on the Location
```

Note that the file "log.txt" no longer exists on the initiating host. Only one copy of the file can exist, and it will exist on the last Location that the mobile agent relocated to.

Increment #5: Relocating Perl Modules

Perl supports a large collection of *third-party extension modules*[3]. If a mobile agent developer uses such an extension within some code, the module may need to be relocated to the Location if the extension is not available at the Location. At the very least, the delivered technology needs a mechanism to check for the existence of an extension module and exit gracefully if the Location will not accept the relocation of the module.

[3] See `http://www.perl.com/CPAN` for more details.

Chapter 4

Design

4.1 Introduction

In the *Design Stage* of any project, the emphasis shifts from the *what* to the *how*. This chapter discusses the *how* in some detail. After describing the development process to be used, a number of proposed approaches to developing the facility are described.

The central mechanism of the facility is support for *arbitrary suspension of a running program, its subsequent relocation to another network device and its resumption there*. This will be referred to as the *Relocation Mechanism*. It has proved difficult to design the relocation mechanism of the facility *up-front*, and the reasons for (and implication of) this are discussed below. Other sub-parts of the facility can be fully designed at this stage, and the design of each is described in detail. These sub-parts are the Security Mechanisms, the Transport Mechanism and the Load-and-Execute Mechanism.

4.2 The Development Process

The *Incremental Model*, as described in [Pressman, 1997], will be used to develop the facility. This software development model combines the classic *Linear Software Development Model* with *Prototyping*, and with each *development cycle* produces an increasingly more functional facility.

Prototyping is generally employed during the *Requirements Phase* of a project, and is used as a tool to *tease out* a set of requirements from a user when traditional requirements analysis is inappropriate (or has failed). When the developed prototype matches (or closely matches) the system which is required, traditional implementation techniques can then be used to design and build the system, or adapt the prototype for real-world use.

When combined with the *Linear Software Development Model*, *Prototyping* can also (perhaps surprisingly) be used during the *Design Phase*. When the requirements are well understood, but it is unclear just how the system is to be developed, a series of small prototypes can be developed in an attempt to determine the best way to design and develop the required facility. Once a winning strategy has been found, successively more functional facilities can be designed and built *incrementally*. In this way, the design evolves, adapts and expands over time.

Typically, when following the *Incremental Model*, each *development cycle* consists of the following steps:

1. **Designing the Increment** - the next piece of the facility is designed.
2. **Developing the Increment** - the increment is coded to reflect the design and implementation decisions made.
3. **Testing the Increment** - the increment is tested to ensure the designed functionality is being delivered.
4. **Integrating the Increment** - the increment is integrated with any existing increments (that is, the facility *evolves*).
5. **Deploying the Increment** - the facility (in its current form) is deployed on the target platform(s), and tested again.

As such, it is not possible to develop a traditional, complete design up-front, as the design will evolve over the lifetime of the project.

4.3 The Relocation Mechanism

Deciding on a development model to adopt for this project was straightforward. However, a central question still remains unanswered: How is a mobile agent facility developed? Specifically, how can a Relocation Mechanism be added to the Perl programming environment? With reference to building such a facility for Perl, there are a number of possible approaches, which include:

1. Modify the Perl Interpreter.
2. Transmit the Perl Environment.
3. Transmit Perl Byte-code.
4. Modify the Mobile Agent Source Code.

Each of these approaches is described and discussed in more detail in the sections which follow.

4.3.1 Modify the Perl Interpreter

As the ongoing development of Perl is an open-source project, the ANSI-C source code to the entire Perl environment is freely and publicly available [Perl5Porters, 2003e]. It is therefore conceivable that the Perl source code could be amended to support this project's facility (for instance, by adding support for the `relocate` command to the list of core commands currently understood by the Perl interpreter).

Just such a strategy was followed by an early version of the *D'Agents* mobile agent system, as described in [Cockayne, 1998]. A source-modified version of the TCL interpreter provides a set of mobile agent commands (or *primitives*), and allows for the development of TCL-based mobile agents. The source-modified TCL interpreter also acts as a Location.

The major advantage of this approach is that, assuming a high competence in the C programming language and enough time, it is likely to succeed. Of course, embarking on any activity which results in a source-modified version of the Perl interpreter is not a trivial undertaking. The main disadvantage is that the new interpreter will be, in effect, no longer Perl, but rather a specialized, mobile agent-enabled dialect of the language. This

immediately places the developed facility at a distinct disadvantage, as seasoned Perl programmers are unlikely to adopt such a version of the language if it means moving away from the established (and standard) version of the interpreter. In addition, as changes are made to the standard Perl source code by the Perl developers[1], incorporating these changes into the new Perl dialect becomes nothing short of a maintenance nightmare. This approach is also non-portable, as there is no guarantee that any source code modifications made on the Linux/UNIX version of Perl will port to (or even compile on) the Windows or Macintosh platforms (which have their own versions of "standard Perl").

Consequently, as a possible approach to developing this project's facility, modifying the Perl source code will only be considered "if all else fails".

A related approach would be to provide for a *distributed fork* system call. When `fork` is invoked, the underlying operating system creates a clone of the current process and continues to execute both the original, parent process and its cloned, child process at the statement immediately after the call to `fork`. If a mechanism could be found to add a distributed version of `fork` to the operating system that could arrange for the cloned child process to continue executing on another, remote network device, a form of mobile agent facility would be available to any process. Such a strategy, although interesting, would require modifying the source code to the underlying operating system. As Linux is the likely deployment platform, such modifications would indeed be possible. However, modifying the Linux kernel source code is not something attempted on a whim, and - no matter how promising an approach this may sound - will not be considered within the scope of this project.

4.3.2 Transmit the Perl Environment

As Perl is an interpreter, this approach proposes transmitting the entire Perl environment from one network device to another. Effectively, this approach would suspend the current Perl process on a network device, transmit it to another network device, then restart the Perl

[1]Known collectively as *The Perl 5 Porters.*

process. Moving processes from network device to network device is a non-trivial activity (as described in [Tanenbaum, 2001], Section 3.4), and is generally regarded as very difficult at best, and all but impossible most of the time.

Like the previous approach, this technique suffers from being a non-portable solution, as a process on Linux is not anything like a process on Windows NT, nor Mac OS 9/X for that matter. In short, this approach is worse than the previous approach and will not be considered further within the context of this project.

4.3.3 Transmit Perl Byte-code

Each time a Perl program is executed, the Perl interpreter scans its source code for errors, converts the source code to Perl's internal byte-code format, optimizes the byte-code, then runs it. The byte-code is what runs, not the source code[2]. If a mechanism can be found to arbitrarily suspend Perl byte-code, a useful facility could be developed assuming another mechanism exists to restart the byte-code from where it left off.

Such an approach is central to *Agent Perl.* The mobile agent's *object byte-code* is captured, then transmitted from network device to network device. However, rather than supporting arbitrary suspension and restart, *Agent Perl* provides standard invocation entry points from where a mobile agent can resume execution once received by a facility. Another shortcoming is that mobile agents based on *Agent Perl* must be implemented as object-oriented derived classes from its main class (called `Agent`), which may not suite all application requirements.

However, as an approach, this technique shows promise. After all, it works for *Agent Perl.* Additionally, the Perl community provides a host of add-on support modules for Perl, including compiler back-ends that can manipulate the Perl byte-code[3]. Only through further research and experimentation will it be possible to determine if Perl's byte-code can be arbitrarily suspended and restarted. Note, too, that (just like Java), Perl's byte-code is

[2] This is very similar to the Java technology, the major difference being that Java deliberately exposes the byte-code to users and developers in the form of `.class` files.

[3] One such example is the `B::*` modules, now included as part of the standard Perl distribution.

designed to be highly portable across platforms.

4.3.4 Modify the Mobile Agent Source Code

The final proposed approach involves "on-the-fly" modification of the source code to the mobile agent itself. As all Perl programs are stored and distributed in source code form, the potential exists to take advantage of this fact by modifying the Perl source code to the mobile agent *while it executes.*

To understand how this approach might work, consider this possible implementation strategy for the proposed `relocate` command. As part of the facility, the `relocate` command, when executed by the Perl interpreter, would:

1. Load a copy of the mobile agent source code (within which the invocation of `relocate` was made) from disk storage into memory.
2. To the copied source code, add a label directly after the line of code that invoked `relocate`.
3. To the copied source code, add another line of code to the start of the source code file to *jump* to the just-inserted label. This line of code would be the first line of code to execute within the modified mobile agent.
4. Send the modified mobile agent source code to its destination.

The destination then simply executes the received mobile agent source code. As the received code has been modified to jump to the instruction immediately following the `relocate` command, the mobile agent continues to execute from where it left off.

As an approach, this implementation strategy is very simple, yet it may prove effective. A downside is that the modified code needs to use the much frowned-upon and maligned `goto` statement (which *is* available to Perl programmers that need it). See [Dijkstra, 1968] for the now classic argument as to why `goto` is considered bad for your programming health. Note that the scoping restrictions placed on the usage of the `goto` statement by the Perl

interpreter may further hamper this approach. However, it is worth noting that this simple mechanism does support *strong mobility*, which is a required feature of the facility.

What this simple example fails to recognize is that, in addition to relocating the code, the facility also has to relocate any state that existed on the initiating network device, where *state* involves the current values of any variables, file-handles or sockets that the mobile agent was using on the initiating network device, immediately prior to suspension and relocation. This is a non-trivial exercise, although a number of Perl add-on modules should help in this area (most notably `Devel::Symdump`, `PadWalker` and `Data::Dumper`).

4.4 The Security Mechanisms

On page 160 of [Tanenbaum, 2001], it states:

> *"Blindly trusting that the downloaded code implements only the advertised interface while accessing your unprotected hard disk and does not send the juiciest parts to heaven-knows-who may not always be such a good idea."*

The provision of the facility introduces a number of security threats, which can be categorized into three main groups.

1. Threats to mobile agents in-transit.
2. Threats from a malicious Location.
3. Threats from a malicious mobile agent.

In each of the sections which follow, discussion is given over to the proposed method of protecting a mobile agent against each of these threats.

4.4.1 Protecting Mobile Agents In-Transit

As a mobile agent travels across the public Internet it is subject to *eavesdropping*, in that an unauthorized third-party may take a copy of the mobile agent in order to learn from the contents of its code and state. *Learning from the code* would allow and eavesdropper to determine the techniques used by the mobile agent in getting its work done. *Learning from the state* would allow an eavesdropper to examine the data carried by the mobile agent. The

classic descriptive example of the problems that could result is to consider a "purchasing" mobile agent that carries credit card information as part of its state. If the mobile agent is compromised, not only is the credit card information revealed but, also, what the mobile agent intended to do with the credit card information is revealed. Obviously, protection is required here.

There are two approaches to securing against this threat. The first is to provide a *secure channel* between each pair of Locations. By ensuring that any transmission occurs on the secure channel, the mobile agent in-transit is protected. However, this solution assumes that the provision of the secure channel is (1) possible and (2) previously agreed to by both parties (i.e., both the sending and receiving Locations). This may not always be the case.

Implementing such a secure channel has been simplified by the widespread adoption and availability of SSH, the Secure SHell, see [Openssh, 2003]. The design pattern is straightforward:

```
determine login-id and password
establish a secure channel using login-id and password
        between the two Locations, A and B
use SCP (Secure CoPy) to transfer the mobile agent
```

The second approach is to encrypt the mobile agent prior to transmission on the public Internet. This can be accomplished with conventional, symmetric encryption or with public-key cryptography[4].

Using conventional cryptographic technologies assumes either the secure transmission of the *shared secret key* or the availability of a trusted third-party (that has access to, and can provide upon request, a shared secret key for use by the sending and receiving Locations). The *Diffie-Hellman Key Exchange Protocol*, first introduced in [Diffee, 1976], can be used to transmit a shared secret key across the insecure public Internet. Using public-key cryptography requires the each Location's *public key* be widely available (and installed on the receiving Location).

For this project, conventional cryptographic techniques will be used to secure the mobile agent in-transit. The *Diffie-Hellman Key Exchange Protocol* will be used to trans-

[4]See [Stallings, 2000b], Chapters 2 and 3 for a good overview of cryptographic techniques.

mit the shared secret key between participating Locations, with the design pattern closely matching the technique described on page 265 of [Singh, 1999]. Note that the use of in-transit encryption will only be employed when the services of SSH are not available to the facility. Of course, all of the above assumes that the receiving Location can be trusted.

4.4.2 Malicious Mobile Agent Environments

Unfortunately, once a mobile agent has been received by a Location it may already be too late to protect the mobile agent if the Location is malicious. The strategy is, therefore, not to attempt to protect the mobile agent in such a situation, but to be able to detect if the mobile agent has been *inappropriately modified* by a malicious mobile agent.

In order to protect mobile agents using the facility, it is proposed to support mechanisms similar to those used by the *Ajanta* system [Ajanta, 2003].

4.4.3 Malicious Mobile Agents

In addition to worrying about malicious Locations, the facility needs to provide for protection from malicious mobile agents. After all, the Location exists to receive *code* from a remote network device and execute it[5]. Mechanisms are required to ensure that the mobile agent only executes those instructions that it is authorized to by the Location. Additionally, the Location needs to protect its host computer from a potentially malicious mobile agent accessing resources that it shouldn't. Such resources could include: hard-disks, printers, monitors and network connections.

In effect, the Location needs to treat any received mobile agents as untrustworthy until such time as the facility can determine that the mobile agent can indeed be trusted. Once trust has been established, the facility needs to decide how much it wants to trust the received mobile agent *before* executing its code.

A common solution is to employ a *sandbox.* This mechanism ensures that code which is executed is controlled, in that only allowed instructions can be invoked by the

[5]The most malicious of mobile agents is known by another name: *computer virus.*

downloaded or received agent. Just such a technique is employed by the Java-applet execution mechanism embedded within most modern web browsers. Within Perl, a standard module called `Safe.pm` provides for just such a facility. `Safe.pm` compiles and executes Perl code within a "restricted compartment".

The facilities of `Safe.pm` will be used by the facility to restrict, where appropriate, the received mobile agent's ability to execute arbitrary code. Only that code which is specifically allowed will execute.

4.5 The Transport Mechanism

As the facility is proposed to operate on the Internet, support for the TCP/IP Suite of Protocols is required. Specifically, the transport services provided by the Transport Control Protocol (TCP) and the User Datagram Protocol (UDP) are to be supported.

Mobile agents built to exploit the facility will be able to choose either TCP or UDP as their transport service. In addition to selecting and using the appropriate protocol, the mobile agent will be able to contact a Location at a specified *location* on the Internet. This location is a predetermined IP address and protocol port-number combination. The design pattern for the Transport Mechanism *when sending* is as follows:

```
determine IP address and protocol port-number of next Location
select between TCP and UDP as a transport service
if TCP selected
    attempt to establish a secure connection with next Location
    if secure connection available
         use SCP to send the mobile agent to the next Location
    else no secure connection available
        attempt to establish a normal connection with next Location
        if connection available
            encrypt the mobile agent and send to next Location
        else (no connection available)
            abort
if UDP selected
    send the mobile agent to next Location
```

Note the following:

1. In determining the IP address and protocol port-number of the next Location, it is

assumed that the mobile agent is manually pre-configured with these data items. It is beyond the scope of the project to provide for a Location naming service.

2. In attempting to establish a secure connection with the next Location, either an SSH secure channel will be established or the *Diffie-Hellman Key Exchange Protocol* will be employed to establish a shared secret key which can then be used to encrypt the mobile agent prior to transmission (as described earlier).

3. The use of UDP as a transport service is provided to primarily support mobile agent relocation within a Local Area Network (LAN). When UDP is employed as the transport service, encryption is not provided. Note, too, that UDP makes no guarantees in relation to ensuring any data entrusted to it is in fact delivered. However, on modern LANs, this UDP limitation is of little concern.

The design pattern for the Transport Mechanism *when receiving* is presented in the next section.

4.6 The Load-and-Execute Mechanism

The Load-and-Execute Mechanism is built into the Location. As the name suggests, this mechanism loads the received mobile agent into memory and executes it (or more precisely, *continues* to execute it). The design pattern for the Load-and-Execute Mechanism is straightforward:

```
wait passively for a mobile agent to arrive
receive mobile agent from another Location
if mobile agent 'damaged'
    abort
create a sandbox within which to run received mobile agent
execute the received mobile agent
```

Note that this rather simple pattern masks the complexity involved in *securely* receiving a mobile agent from another Location.

Chapter 5

Implementing the Facility

5.1 Introduction

This chapter discusses the *Implementation Phase* of this project. Each of the four central mechanisms required by this project (Relocation, Security, Transport and Load-and-Execute) were developed *in parallel*. Obviously, in order to provide for the entire facility, all four mechanisms needed to exist and work together seamlessly. However, during the *Development Phase*, each was worked on separately.

For instance, strange though it may seem, the Load-and-Execute Mechanism does not depend on the existence of a working Transport Mechanism, it just needs to know where on the Location's local storage it should load the received mobile agent from before continuing to execute it. Equally, the Transport Mechanism was initially developed without any of the services provided by the Security Mechanism being in place. The "thing" to be transported from one Location to another results from the Relocation Mechanism, but the Transport Mechanism can treat the "thing" to be transported as a file, so it did not matter (initially) that the file is or is not a mobile agent.

The initial implementation strategy was to quickly develop a prototype Relocation Mechanism based on the technique outlined in the last chapter, namely the *Modify The Mobile Agent Source Code* technique. It was not envisaged that a facility built on this

technique would provide the "ultimate solution". However, it was easy to implement, and did allow for a limited, working implementation that was used to support the development and testing of the Security Mechanism. If, after the required experimentation, progress could be made on building a Relocation Mechanism that could transport Perl byte-code[1], the Security Mechanism was already be in place.

As detailed in [Barry, 2002], [Stein, 2001] and [Stevens, 1998], the techniques used when designing and developing a Transport Mechanism are well understood. Consequently, the creation of a working Transport Mechanism for this facility was straightforward.

Developing the Security Mechanisms was a challenge, but, the technologies around which the Security Mechanisms were built are well understood (e.g., Diffie-Hellman Key Exchange and SSH) and documented elsewhere. In many cases, public implementations were also available (e.g., The Ajanta System).

5.2 Solving the Central Problem

The central question to be answered by the facility is:

> *How can a Relocation Mechanism be added to the Perl programming environment?*

The initially selected Relocation Mechanism is simple and somewhat crude. It involves determining the line of the source code that requests relocation, then amending the source code so that the relocated code would jump to the instruction immediately following the *relocation request*.

Amending the source code is a simple text-processing exercise. The "crude" part involves the use of the much-maligned `goto` statement to perform the jump. To illustrate this approach, consider this small sample program:

```
print "On original location.\n";
relocate( 'testimac.itcarlow.ie', 2001 );
print "On next location.\n";
```

The Relocation Mechanism transforms this source code into the following:

[1] Which was the *preferred* solution.

```
goto Label_to_jump_to;

    print "On original location.\n";
    relocate( 'testimac.itcarlow.ie', 2001 );
Label_to_jump_to:
    print "On next location.\n";
```

In effect, the source code *mutates* as a result of relocating. The "*On original location.*" message appears on the screen of the initial computer, and the "*On next location.*" message appears on the screen of the computer known as `testimac.itcarlow.ie`. Simple, crude, but nonetheless, and as will be shown, effective. Implementing this approach led to another unanswered question:

> *Without having to resort to changing the source code to the Perl interpreter, how can the interpreter be told what to do when it encounters the* **`relocate`** *invocation?*

That is, how is it possible to stop a program "in its tracks", then start it again? A technology already exists to do just this: the source code debugger. In fact, *every* debugger is capable of stopping a running program, then starting it again. Additionally, most every debugger is capable of examining the state of the running program and - if necessary - adjusting the state if needs be. Some debuggers (including Perl's, as described in [Perl5Porters, 2003a] and [Scott, 2001]) provide a mechanism to *insert code* into the running process and, consequently, *mutate* the program as it executes. So, to build the Relocation Mechanism would require the development of a custom debugger. And this is exactly what the facility is: a debugger. As such, the facility is capable of starting, stopping and restarting a program on demand. It is also capable of examining the state of an executing program and mutating its source code.

The developed debugger is implemented in the `Scooby.pm` module[2]. The Relocation Mechanism is triggered by the use of a small module called `Executive.pm` which is imported at the start of a mobile agent's source code.

As discussed in the last chapter, the Load-and-Execute Mechanism is straightforward. Within the facility, this mechanism is implemented in a module called `Location.pm`.

[2] And is christened "Scooby" for want of a better name.

This module implements an object-oriented class that can be used to develop Locations. This class provides the facility with the ability to receive a mobile agent from a remote Location, and re-execute it.

The implementation of the Transport Mechanism is integrated into the `Scooby.pm` module's source code, as well as the Location building module, `Location.pm`.

The developed Security Mechanism provides an environment within which a Location can verify the integrity of any source code received from a mobile agent operating on another computer. The Location can then decide if it wishes to execute the received source code based on its ability to authenticate. The mobile agent ensures that the sent source code can only be of use to a single Location by encrypting the source code in such as way that decryption can only be successfully performed by the correct Location. Consequently, if the encrypted source code is tampered with en-route, or redirected to an incorrect Location, the Security Mechanism can take any appropriate action to ensure that the mobile agent is aborted or rejected.

A public-key cryptosystem based on the RSA technologies, see [RSA, 2003], supports the operation of the Security Mechanism. Each entity within the system (i.e., each mobile agent and each Location) has both a RSA Public Key and RSA Private Key. By digitally signing a message with an RSA Private Key prior to delivery, a recipient can use the corresponding RSA Public Key to check the integrity of the message, as only the owner of the Private Key that corresponds to the Public Key could have possibly signed it[3]. By encrypting a message with a recipient's RSA Public Key, only the recipient can successfully decrypt the message and determine the original contents, as only the recipient knows its own RSA Private Key.

The facility takes advantage of these behaviors to implement the Security Mechanism in the `Scooby.pm`, `Executive.pm` and `Location.pm` modules.

Despite all of the obvious advantages, public-key cryptosystems have to satisfactorily answer one "sticky" question:

[3]Assuming, of course, that the Private Key has not been compromised.

> *How do players within the facility convince themselves that the RSA Public Key for the entity they wish to communicate with is in actual fact the RSA Public Key for that entity, and not some forgery?*

The facility answers this question by providing, as an integral component of the facility, a Key Server. The Key Server's role is to provide a secure repository for RSA Public Keys. Prior to any secure communication, the Key Server is contacted to provide the correct RSA Public Key to use. All communications to/from the Key Server are themselves secure, and they too employ the RSA public-key cryptosystem. Any data on the facility's RSA Public Keys is persistently maintained in an MySQL RDBMS (see [Dubios, 1999] for a good overview of MySQL).

5.2.1 A Matter of Trust

The Key Server is designed to operate on a secure computer within the domain of the facility. It is not recommended that the Key Server operate on a computer that hosts a Location or provides for the relocation of mobile agents. For the Key Server to function at all, the other entities have to trust the information that it provides and that the Key Server is itself not compromised. There are a number of well-know "system-hardening" techniques available to systems administrators today. However, when all is said and done, everything boils down to trust. It is not possible to produce a totally automated authentication/encryption system. Sooner or later, you have to trust somebody (or something). And within the facility, the Key Server is *assumed* to be trusted. The Key Server provides three services to clients that access it:

1. **Registration** - allows a client to register an RSA Public Key within the Key Server database. Three pieces of information are added to the `SCOOBY.publics` database table: the IP address and protocol port number of the mobile agent, and the RSA Public Key (in `Crypt::RSA` disk-file format). The client supplies the RSA Public Key and the protocol port number, while the Key Server deduces the IP address from the client's Socket connection.

2. **Responding** - allows a client to lookup an RSA Public Key within the Key Server database. The client provides an IP address and protocol port number combination, and the Key Server responds with the RSA Public Key associated with them.

3. **Monitoring** - allows any web browser to view the status log-file maintained by the Key Server. See the section below for more details on this service (which is also available from each Location).

5.2.2 A Mutating Automatic Debugger

As discussed earlier, the `Scooby.pm` module provided as part of the facility is a debugger. Unlike traditional debuggers, the facility does not interact with the programmer. Instead, the facility automatically interacts with the mobile agent that is currently executing.

The Perl interpreter provides a user-programmable series of hooks into its debugging mechanism, as described in [Perl5Porters, 2003b]. Two of these hooks are exploited by the facility

The first, the *DB hook*, is invoked for every statement in a program that can be "breakpointed". Within the *DB hook*, the value of the current name-space, filename and line number are accessible. The latter two values are of particular interest to the facility

The second hook, the *sub hook*, is invoked immediately prior to a subroutine call. In addition to having access to the name of the subroutine called, the *sub hook* also has access to the values passed as parameters to the called subroutine. The *sub hook* can preform any amount of preprocessing before (optionally) deciding to invoke the called subroutine. The *sub hook* is used by the facility to determine when the mobile agent has invoked the `relocate` command.

When `relocate` is invoked, the facility replaces the mobile agent's call to `relocate` with its own implementation, which performs a series of actions, as follows:

1. Examines the state of the mobile agent and identifies the values associated with any lexical variables.

2. Mutates the mobile agent source code to include a collection of Perl statements that will reinitialize the values of any lexical variables once the mobile agent is next executed.

3. Contacts the Key Server to determine the RSA public key to use when encrypting the mobile agent prior to relocation. The RSA public key requested is that of the next Location to relocate to.

4. Encrypts the mutated mobile agent source code with the RSA public key of the next Location, producing cyphertext.

5. Digitally signs the cyphertext with the mobile agent's RSA private key.

6. Sends the name of the mobile agent together with the number of the next line to execute to the next Location.

7. Registers a copy of the mobile agent's RSA public key with the Key Server.

8. Sends the digital signature together with the cyphertext to the next Location.

At the next Location, the facility verifies the digital signature against the cyphertext. If the verify is successful, the cyphertext is decrypted with the Location's RSA private key. The mutated mobile agent source code is reconstituted on the Location then mutated again (by the Location) to ensure that the next time it is executed it will continue from the statement immediately following the most recent `relocate` invocation. The next section describes this process by way of example.

5.2.3 What Happens: Step-by-Step

To illustrate what happens to a mobile agent when executed by the facility, let's consider the following simple program:

```
#! /usr/bin/perl -w

# exampleagent.pl - a simple, example mobile agent.
```

```
use Mobile::Executive;

my $a = 10;
my $b = 5;
my $c = 2;

relocate( 'pblinux.itcarlow.ie', 2001 );

my $result = ($a + $b) * $c;

print "The result of the calculation is: $result\n";
```

This program defines three scalar variables (`$a`, `$b`, and `$c`), then relocates to the Location running at protocol port number 2001 on the machine identified by `pblinux.itcarlow.ie`. Once there, it performs a calculation and assigns the result to another scalar variable, called `$result`. The result is then printed to the screen with an appropriate message.

Prior to relocation, the facility (in the `Scooby.pm` module) mutates this program's source code to include the statements required to reinitialize any lexical variables that exist prior to the invocation of the `relocate` command. After this mutation, the source code looks like this:

```
#! /usr/bin/perl -w

# exampleagent.pl - a simple, example mobile agent.

use Mobile::Executive;

my $a = 10;
my $b = 5;
my $c = 2;

relocate( 'pblinux.itcarlow.ie', 2001 );
    $a = 10;
    $b = 5;
    $c = 2;

my $result = ($a + $b) * $c;

print "The result of the calculation is: $result.\n";
```

The mutation has added three assignment statements immediately after the call to `relocate`. The mutated source code is then encrypted, producing the following cyphertext:

```
-----BEGIN COMPRESSED RSA ENCRYPTED MESSAGE-----
Scheme: Crypt::RSA::ES::OAEP
Version: 1.24
```

```
eJwBEQLu/TEwADUxMgBDeXBoZXJ0ZXh0wR30an5qW3IGwaJlb02LPlt42j6M66hy2Em8qsJa8uSd
qcNs58m5iP06zavqjKZYeorruTxakVfwifGGKNavZ79nZQ2Xnc0E3bCRa9iULsV1XZbbLfneV455
0iXBxAA4jEHpgCsRiBqV4RfneSsLp+W18a8B+n9AZJphc0zU9AS3gq51UtMiJEYg8wGqGQS+TwcV
OR4IOVNOjlHruR5Qs9OCR+sA8761k74cpRmsbvQ9qigJPOdohtKi0p9lD3Ixw0oTB6aEGVW+Joc9
LVOsHTgCkXfY2skKerN7vGYqsk674yeTVeK42+EldeLL3RSlUVfgTlDtbmQ/FvVMesWOKyDnS6el
Mjd79TDhUOTCef3vhO/blfH8xFKlUj58QWg4zaKzJn3sQGioF/GGgFnhe2QPJ/tUvQFauKWGpUDR
c+7OO9J6UcCAnIcgYdVel0hsYL57KsXRuUV+stoPXNozIImM/8e9OlURYFQ+goIUm9bLC3xzHbg4
7oBJM+vCgYehSaa1VpSFUW4KFqcScTN0tzBfvEBCxE3s6sLzuZwc1G5HNN72pPV7quE5m9IwQHkU
y3cdP7Ersht0EzW8wzZzJKfr4KKyr49EKmswStqw7P9ehSXWcud4yLN1Idl+ihWfd29FoomJZSNi
B1VR/2SvH44ZXbwlcqz2Y/ezzlkalwDdjQjn
=e+gkL50R4aBCb2AiMkMhRg==
-----END COMPRESSED RSA ENCRYPTED MESSAGE-----
```


The cyphertext is then digitally signed, producing this signature:


```
-----BEGIN RSA SIGNATURE-----
Scheme: Crypt::RSA::SS::PSS
Version: 1.5

0QAxMjgAU2lnbmF0dXJldZdVsD6lAxKbwhStAV7ZKpRpkw8optUpeerWPGPmZq3/xA4TCQMw/bR0
jUD5nn8VKffIJrA5JZWbpnK0VGNpOOeexkzUBZWJX1598owemIjHhsts+uzTStUAXlvV0lWbnDx6
u0l2VD0uGq3c37PPbkijRRJplNLqqwGa3/qwEuk=
=iaLiyw2NxLnzcMNE/Vggtg==
-----END RSA SIGNATURE-----
```


At this point, the facility (again within the `Scooby.pm` module) sends the digital signature and the cyphertext to the next Location which, in this case, is running on `pblinux`. Once there, the digital signature is verified to be correct prior to the decryption of the cyphertext, which produces the mutated source code. This source code is then mutated by the Location running on `pblinux` to jump to the instruction immediately following the invocation of `relocate`, producing this mutated source code:

```
#! /usr/bin/perl -w
goto LABEL_exampleagentpl121053705383;

# exampleagent.pl - a simple, example mobile agent.

use Mobile::Executive;

my $a = 10;
my $b = 5;
my $c = 2;

relocate( 'pblinux.itcarlow.ie', 2001 );
LABEL_exampleagentpl121053705383:
1;
use Mobile::Executive;
```

```
    $a = 10;
    $b = 5;
    $c = 2;

my $result = ($a + $b) * $c;

print "The result of the calculation is: $result.\n";
```

Note the addition of a `goto` statement immediately after this programs first line[4]. The label to jump to has also been inserted immediately after the call to `relocate`, thus facilitating the re-execution of this program from where it last left off. The original source code has mutated twice and relocated once. When executed by the Location running at `pblinux.itcarlow.ie`, the following message appears on screen:

```
The result of the calculation is: 30.
```

As expected, the actual calculation is performed on the relocated-to Location. Which is all somewhat crude, but effective all the same.

5.2.4 The Web-based Monitoring Services

Both the delivered Key Server and the `Location.pm` module provide support for remote monitoring of their activity via the world-wide-web. Upon startup (and by default), both technologies start a HTTP server on protocol port number 8080. When contacted by any web browser, these servers provide a simple log-viewing facility. In effect, they allow the administrator of the facility to check on the status of the Key Server or any Location without having to physically visit the hosting computer. Examples of the data available to the user of this service are included in *The Scooby Guide*, a copy of which is included on the accompanying CD.

5.3 Testing and Debugging

As each increment was built, and as the functionality of the facility was extended, unit testing was performed on each component. This testing was supplemented as required. For

[4]Often referred to as the “shebang” line, as it starts with the UNIX-specific `#!` sequence.

example, a collection of small test drivers were developed to exercise the Key Server, and the *MySQL Monitor* [Widenius, 2002] was used to check that the `SCOOBY.publics` database was in the correct state.

As soon as the components of the facility matured to the degree were they could be integrated, system testing was complicated by a number of factors, which included:

Distributed Platform Problems - As the facility operated within a distributed environment, tracking down bugs was often difficult. Testing and debugging on a single computer is tough enough, and was only complicated by the fact that the major components of the facility were executing on (at least) three different computers. Consequently, when bugs appeared, it was hard to determine within which part of the facility the bugs were actually originating.

No Debugger - The fact that `Scooby.pm` is a debugger meant it was not possible to debug it with the Perl Debugger, as only one debugger can be active at any time. This was frustrating. Bug-tracking relied heavily on the use of "print" statements scattered throughout the source code, which provided a basic trace of the execution. In addition to being primitive, this debugging technique was very time consuming.

The addition of the web-based logging mechanisms to the `Location.pm` module and the Key Server program helped with debugging.

Confirming that the messages passed around the network were correct and as expected was accomplished by the use of the *Network Debugger* from [Barry, 2002]. This tool allowed for the capturing of the traffic generated by each of the networked devices comprising the facility, in effect documenting the flow of execution through the network.

In the following sections we present a diagram of the test network, and consider a serious "bug" that presented itself late in the development process. It is described here not only because it is interesting, but also because it is representative of the type of problems that can occur within a complex distributed system. Finally, we present a number of network traces captured during the testing phase. These traces are supplemented (where appropriate)

with output produced by the *MySQL Monitor*.

5.3.1 The Test Network

The network employed during the development of the facility resides at *The Institute of Technology, Carlow*. The networked devices used are identified on the network diagram on page 48, together with the LAN segments they are attached to[5].

Those network devices sporting an 'R' identify routers running proprietary "network" operating systems. All of the other network devices are running some version of the Linux Operating System. A square denotes a client (or workstation) network device, whereas a rectangle denotes a server-class network device.

On this diagram, `glasnost` is running the Key Server, whereas `pblinux`, `pbmac` and `testimac` are all running as Locations. The `tyndall` computer provides name services to the LAN, and `gw` connects the LAN to the global Internet.

During testing, `pbmac` ran a Location at protocol port number 2001, `testimac` ran at protocol port number 3001 and `pblinux` ran at protocol port number 4001.

5.3.2 The Ordering of Events within a Distributed System

Recall that the Key Server provides three services: Registration, Responding and Web-based Monitoring. When a mobile agent attempts to relocate (with some help from the `Scooby.pm` module), it first registers the its RSA Public Key with the Key Server. The mobile agent also requests a copy of the next Location's RSA Public Key (from the Responding Service). The mobile agent can then encrypt its source code with the next Location's RSA Public Key before digitally signing a copy of the cyphertext with its own RSA Private Key. Sending the RSA Public Key to the Key Server is *critical*, as the next Location needs the key in order to verify the cyphertext, in effect allowing for the reconstitution of the mobile agent's source code. The sequence of events is this:

1. The mobile agent registers its RSA Public Key with the Key Server.

[5]The diagram is based on a similar diagram from [Barry, 2002].

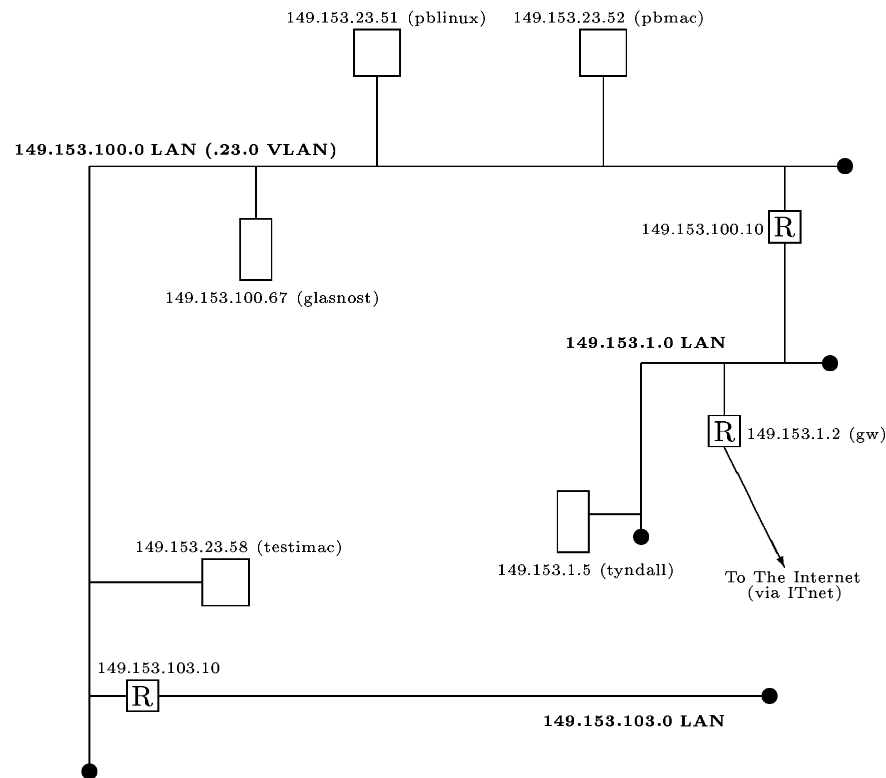

Figure 5.1: The Test Network at The Institute of Technology, Carlow.

2. The mobile agent encrypts its source code, producing the cyphertext.

3. The mobile agent digitally signs the cyphertext, producing the signature.

4. The mobile agent sends the signature/cyphertext combination to the Next Location.

5. The next Location receives the signature/cyphertext combination.

6. The next Location asks the Key Server to provide the mobile agent's RSA Public Key.

7. Using this RSA Public Key, the next Location verifies the signature, then (assuming success), decrypts the cyphertext, producing the mobile agent's source code.

This sequence works, assuming the events occur *in this order*. Within a distributed computing environment, events are often not guaranteed to occur in any particular order, as described in [Tanenbaum, 2001]. The fact that the Key Server is built around a concurrent,

multi-process design only adds to the coordination complexity.

When the first version of the facility was developed, the next Location kept reporting that it could not verify the received digital signature. Upon closer inspection, it transpired that the Key Server was reporting `NOSIG/NOTFOUND` when the next Location requested the RSA Public Key for the just received mobile agent. This was perplexing.

How could this occur, especially when the mobile agent had registered the RSA Public Key *prior* to sending the digital signature to the next Location?

For some reason, the registration of the mobile agent's RSA Public Key was not occurring fast enough. Why was the Key Server accepting the registration but not making it immediately available to the Responding Service?

Attempts to solve this problem initially concentrated on the sending side. That is, the source code within the mobile agent's relocation mechanism (within `Scooby.pm`) was examined to ensure it was operating correctly. It was working properly, in that the RSA Public Key was being sent to the Key Server. The source code that implemented the receiving side was then examined, within the `Location.pm` class. It too was operating correctly. This led to the conclusion that the problem must be *between* the sending and receiving side. Suspicion was redirected on the employed transport protocol, TCP, the Transmission Control Protocol.

Although it is a reliable transport protocol, TCP does not maintain "record boundaries". In essence, this means that TCPs internal buffering can sometimes interfere with how quickly data is delivered. For instance, what is sent as five short 1K messages on the sending side of a TCP connection may be delivered as 1 large 5K message on the receiving side. The reasoning behind this behaviour is that if it is more efficient for TCP to send one large chunk of data as opposed to five small ones, then that is what TCP will do "automatically", without informing the sending side. As no attempt is made to maintain record boundaries, TCP concerns itself solely with the reliable delivery of the data entrusted to it.

Could this behaviour be causing the problem, in that the sending side is convinced the data has been sent, but the TCP buffering employed is delaying its delivery to the Key

Server, resulting - ultimately - in the `NOSIG/NOTFOUND` message on the receiving side?

Additional programming constructs were added to force TCP on the sending side to deliver its data as soon as data was written to TCP, effectively turning off TCPs default behaviour and turning on record boundaries. This change made no difference.

Source code was then added to the sending side to request the RSA Public Key from the Key Server immediately after registering it. By requesting an acknowledgment from the Key Server *before* sending the digitally signed cyphertext to the next Location, the sending side could be assured that the Key Server was responding correctly.

Somewhat unexpectedly, the sending side registered the RSA Public Key with the Key Server, then requested the very same RSA Public Key from the Key Server, only to receive the `NOSIG/NOTFOUND` message from the Key Server! The RSA Public Key had just been registered, but was not yet available to the Responding Service. What was going on?

Suspicion shifted to the Key Server. Recall that one process within the Key Server supports registration, while another responds to lookup requests. The latter process simply performs an appropriately formed SQL SELECT query against the `SCOOBY.publics` table, returning the RSA Public Key found or the `NOSIG/NOTFOUND` message if the RSA Public Key is not found. By contrast, the registration service uses a similarly formed SQL SELECT to first determine whether or not the RSA Public Key already exists in the `SCOOBY.publics` table, then performs an SQL UPDATE query if it does, or an SQL INSERT if it does not.

The SQL SELECT/UPDATE (or SELECT/INSERT) query combination (from the Registration Service) obviously takes much longer that the sole SQL SELECT query (from the Responding Service). As a result, the Registration Service cannot update the `SCOOBY.publics` table with a new RSA Public Key within a time-frame that allows the Responding Service to report the same RSA Public Key as existing. And this explains why the facility was reporting the `NOSIG/NOTFOUND` message under these circumstances.

The solution to the problem was not to change the Key Server. The solution amended the sending side to *wait for the Key Server* to respond with the RSA Public Key that has just been sent to the Key Server (and that the sending side knows it has

registered). Only then can the sending side transmit the digitally signed cyphertext to the next Location safe in the knowledge that the next Location can verify, then decrypt it.

This "bug" took five days of concerted effort to diagnose and fix, and clearly demonstrates the type of problem that can occur within distributed computing environments.

This particular problem was now fixed, but not totally. A rather subtle problem remains. To understand this subtle problem, it is necessary to look at the process that the sending side goes through when registering with the Key Server. Here are the steps:

1. The RSA Public Key is generated.
2. The sending side determines the protocol port number that it is using.
3. The RSA Public Key and the protocol port number are registered with the Key Server.

Note that the protocol port number is dynamically allocated to the sending side by the underlying operating system. The range of possible numbers is from 1024 through 49151, as described in [IANA, 2003]. Each time a process communicates over the Socket API, the operating system allocates the next unused protocol port number from this range.

Consider this scenario: a mobile agent is allocated 1026 as its protocol port number on the sending side (the assumption is that the computer hosting the mobile agent has just started operating and only a small number of protocol port numbers have been allocated). This value is then used (together with the RSA Public Key) to register the mobile agent with the Key Server. The computer hosting this particular mobile agent is then restarted. When the computer reboots, the available range of protocol port numbers again starts from 1024. After some communication, a second mobile agent prepares to relocate. The RSA Public Key is generated and the protocol port number is determined. The second mobile agent is also allocated a protocol port number of 1026. The two values are registered with the Key Server.

What happens next is the crux of the issue. The sending side requests an acknowledgment from the Key Server that the RSA Public Key exists for protocol port number

1026. The Key Server, finding the RSA Public Key from the *previous* mobile agent confirms that a RSA Public Key does exist. However, we know from the problem described earlier that the RSA Public Key returned cannot be the most recent - it is the RSA Public Key from the previous mobile agent, and is now out-of-date. Unwittingly, the sending side proceeds to send the cyphertext to the next Location, the next Location requests the RSA Public Key from the Key Server, receives the out-of-date RSA Public Key and fails to verify the current mobile agent's signature. Which is, if truth be told, somewhat of a disaster.

Despite this scenario having the potential to occur, extensive testing has shown that it is highly unlikely to present in practice. The solution to this issue is to amend the Key Server and the sending side to include the RSA Public Key within the SQL SELECT query used during acknowledgment.

5.3.3 Sample Network Traces

This section presents a number of network packet captures (or *traces*) generated by the facility during a simple relocation. Note that, due to space considerations, only the critical parts of the network traces are shown.

Prior to starting the Key Server for the first time, the *MySQL Monitor* is used to confirm that the `SCOOBY.publics` table is empty:

```
mysql -u perlagent -p SCOOBY
Enter password: ************
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 3.23.56

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> select * from publics;
Empty set (0.00 sec)
```

The Key Server is then started, and it generates the following screen messages:

```
Accepting connections/requests from:
  -> 149.153.23.51 on port(s): *.
  -> 149.153.23.58 on port(s): *.
  -> 149.153.23.52 on port(s): *.
```

```
Generating a public/private key-pairing for this keyserver.  Please wait ...
Generated.  Keyserver starting ...
The Responder Service is starting up on port: 30001
The Registration Service is starting up on port: 30002
```

The `SCOOBY.publics` database table is again queried to confirm the addition of the Key Server's RSA Public Key (note: this output has been reformatted so as to fit on the printed page):

```
mysql> select * from publics;
+------------+---------------+-------------------------------------------------+
| ip_address | protocol_port | public_key                                      |
+------------+---------------+-------------------------------------------------+
| 127.0.0.1  | 30001         | $VAR1 = bless( {                                |
|            |               |                 'e' => 65537,                   |
|            |               |                 'n' => '104720326656475888226897 |
|            |               |                         095089929407557867689709 |
|            |               |                         680718550510964913667237 |
|            |               |                         580764386181022399767582 |
|            |               |                         230078520991168085037348 |
|            |               |                         077695823212646956345884 |
|            |               |                         600971189534164028743885 |
|            |               |                         567308638615730739799838 |
|            |               |                         281604196986106438121997 |
|            |               |                         994526991663949945686368 |
|            |               |                         510574399628174303028348 |
|            |               |                         821542543357664583807795 |
|            |               |                         847762499525637106879',  |
|            |               |                 'Version' => '1.91',             |
|            |               |                 'Identity' => 'Scooby Key Server'|
|            |               |                 }, 'Crypt::RSA::Key::Public' );  |
+------------+---------------+-------------------------------------------------+
1 row in set (0.00 sec)
```

A Newly Created Location Registering with the Key Server

With the Key Server up-and-running, a Location is started on `pbmac` (with IP address 149.153.23.52), running on protocol port number 2001. The following network traffic was generated between the Location and the Key Server running on `glasnost` (with IP address: 149.153.100.67). After the connection is established, the Location sends both the protocol port number (2001) and the generated RSA Public Key to the Key Server, before closing the connection[6]:

[6] Note that connection establishment and tear-down within TCP requires three distinct messages, and that every sent message is acknowledged. These facts account for the "extra" messages (or segments) captured by the *Network Debugger*.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 16960, ttl: 64)

TCP Source: 1298 -> TCP Destination: 30002
TCP Header Length: 10, TCP Checksum: 11404
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.52 (id: 0, ttl: 63)

TCP Source: 30002 -> TCP Destination: 1298
TCP Header Length: 10, TCP Checksum: 4235
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 16961, ttl: 64)

TCP Source: 1298 -> TCP Destination: 30002
TCP Header Length: 8, TCP Checksum: 16160
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 16962, ttl: 64)

TCP Source: 1298 -> TCP Destination: 30002
TCP Header Length: 8, TCP Checksum: 53937
TCP Data:

2001

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.52 (id: 25537, ttl: 63)

TCP Source: 30002 -> TCP Destination: 1298
TCP Header Length: 8, TCP Checksum: 16203
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 16963, ttl: 64)

TCP Source: 1298 -> TCP Destination: 30002
TCP Header Length: 8, TCP Checksum: 1676
TCP Data:

$VAR1 = bless( {
                 'e' => 65537,
                 'n' => '941048031542635900439428042509705154394493245476969
                         520809208899834359499866540014249642846160335428535
                         127178488810797552383541159933203055514287799971769
                         261191103264704781411096969000696004933915587139843
                         826185961828581046723030003967554128713593496768476
                         668064354013962005779386497391732057384648648956041
                         13',
                 'Version' => '1.91',
                 'Identity' => '149.153.23.52:2001 Location'
               }, 'Crypt::RSA::Key::Public' );

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 16964, ttl: 64)

TCP Source: 1298 -> TCP Destination: 30002
TCP Header Length: 8, TCP Checksum: 15622
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.52 (id: 25538, ttl: 63)
```

```
TCP Source: 30002 -> TCP Destination: 1298
TCP Header Length: 8, TCP Checksum: 15031
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.52 (id: 25539, ttl: 63)

TCP Source: 30002 -> TCP Destination: 1298
TCP Header Length: 8, TCP Checksum: 15028
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 16965, ttl: 64)

TCP Source: 1298 -> TCP Destination: 30002
TCP Header Length: 8, TCP Checksum: 15620
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The `SCOOBY.publics` database table has been updated with the RSA Public Key for the newly registered Location. Note the the database table contains the RSA Public Key as previously sent from the Location to the Key Server:

```
mysql> select ip_address, protocol_port from publics;
+---------------+---------------+
| ip_address    | protocol_port |
+---------------+---------------+
| 127.0.0.1     | 30001         |
| 149.153.23.52 | 2001          |
+---------------+---------------+
2 rows in set (0.00 sec)

mysql> select public_key from publics where ip_address = '149.153.23.52';
+--------------------------------------------------------------------------+
| public_key                                                               |
+--------------------------------------------------------------------------+
| $VAR1 = bless( {                                                         |
|                 'e' => 65537,                                            |
|                 'n' => '941048031542635900439428042509705154394493245476969  |
|                         520809208899834359499866540014249642846160335428535  |
|                         127178488810797552383541159933203055514287799971769  |
|                         261191103264704781411096969000696004933915587139843  |
|                         826185961828581046723030003967554128713593496768476  |
|                         668064354013962005779386497391732057384648648956041  |
|                         13',                                             |
|                 'Version' => '1.91',                                     |
|                 'Identity' => '149.153.23.52:2001 Location'              |
|                }, 'Crypt::RSA::Key::Public' );                           |
+--------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

With the Location up-and-running and registered with the Key Server, a mobile agent can now relocate to the Location. The following test program will be used throughout the remainder of this section:

```
#! /usr/bin/perl -w

# exampleagent.pl - a simple, example mobile agent.

use Mobile::Executive;

my $a = 10;
my $b = 5;
my $c = 2;

relocate( 'pbmac.itcarlow.ie', 2001 );

my $result = ($a + $b) * $c;

print "The result of the calculation is: $result\n";
```

This example mobile agent starts executing on `pblinux`, then relocates to the next Location running at protocol port number 2001.

A Mobile Agent Requesting Public Keys from the Key Server

The mobile agent queries the Key Server for the RSA Public Key of the next Location. The mobile agent (149.153.23.51) starts by establishing a connection with the Key Server (149.153.100.67):

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 40855, ttl: 64)

TCP Source: 1252 -> TCP Destination: 30001
TCP Header Length: 10, TCP Checksum: 60532
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 0, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1252
TCP Header Length: 10, TCP Checksum: 45382
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 40856, ttl: 64)

TCP Source: 1252 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 57307
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The mobile agent then requests the RSA Public Key of the Key Server, which is associated with IP address 127.0.0.1 and protocol port number 30001:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 40857, ttl: 64)

TCP Source: 1252 -> TCP Destination: 30001
```

```
TCP Header Length: 8, TCP Checksum: 58882
TCP Data:

127.0.0.1

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 2144, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1252
TCP Header Length: 8, TCP Checksum: 57345
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 40858, ttl: 64)

TCP Source: 1252 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 19299
TCP Data:

30001
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 2145, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1252
TCP Header Length: 8, TCP Checksum: 57339
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The Key Server sends its own RSA Public Key to the mobile agent (note the "`SELFSIG`" token within the packet capture):

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 2146, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1252
TCP Header Length: 8, TCP Checksum: 42898
TCP Data:

SELFSIG
--end-sig--

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 26334, ttl: 63)

TCP Source: 22 -> TCP Destination: 1084
TCP Header Length: 8, TCP Checksum: 54750
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 58061, ttl: 64)

TCP Source: 1084 -> TCP Destination: 22
TCP Header Length: 8, TCP Checksum: 12648
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 2147, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1252
TCP Header Length: 8, TCP Checksum: 48020
TCP Data:
```

```
$VAR1 = bless( {
                 'e' => 65537,
                 'n' => '1047203266564758882268970950899294075578676897096
                         8071855051096491366723758076438618102239976758223
                         0078520991168085037348077695823212646956345884600
                         9711895341640287438855673086386157307397998382816
                         0419698610643812199799452699166394994568636851057
                         4399628174303028348821542543357664583807795847762
                         499525637106879',
                 'Version' => '1.91',
                 'Identity' => 'Scooby Key Server'
               }, 'Crypt::RSA::Key::Public' );

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 40860, ttl: 64)

TCP Source: 1252 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 57264
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The mobile agent then requests the RSA Public Key of the next Location, which is running on protocol port number 2001 on `pbmac` (which has an IP address of 149.153.23.52):

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 56800, ttl: 64)

TCP Source: 1253 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 16632
TCP Data:

149.153.23.52

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 35235, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1253
TCP Header Length: 8, TCP Checksum: 41576
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 56801, ttl: 64)

TCP Source: 1253 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 16331
TCP Data:

2001
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 35236, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1253
TCP Header Length: 8, TCP Checksum: 41572
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

And, once again, the Key Server obliges by sending the digitally signed RSA Public Key for the Location:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 35238, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1253
TCP Header Length: 8, TCP Checksum: 63998
TCP Data:

-----BEGIN RSA SIGNATURE-----
Version: 1.5
Scheme: Crypt::RSA::SS::PSS

0QAxMjgAU2lnbmF0dXJlNhIcvTwW4eOp8ufUurTjVmSGWR8p+VvugJKc3CPtKzU2FucHl5bgZiUd
2zFysNFQ6vGUxPFCx8d/FgN5zE8qwdpat3uRb0qfoS8dn6AgTdkQn+tLAjN83/ejKOow7aLfh8Zb
qk1VJVvo02t6Wz/UJDHaI2qKTnMZphExCVcLtGI=
=C4kTHCdaOWQytJ+sM1q7Xw==
-----END RSA SIGNATURE-----

--end-sig--

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 26338, ttl: 63)

TCP Source: 22 -> TCP Destination: 1084
TCP Header Length: 8, TCP Checksum: 31325
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 58065, ttl: 64)

TCP Source: 1084 -> TCP Destination: 22
TCP Header Length: 8, TCP Checksum: 12172
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 35239, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1253
TCP Header Length: 8, TCP Checksum: 60981
TCP Data:

$VAR1 = bless( {
                 'e' => 65537,
                 'n' => '9410480315426359004394280425097051543944932454476
                        96952080920889983435949986654001424964284616033
                        42853512717848881079755238354115993320305551428
                        79997176926119110326470478141109696900069600493
                        91558713984382618596182858104672303000396755412
                        71359349676847666806435401396200577938649739173
                        0573846486489560413',
                 'Version' => '1.91',
                 'Identity' => '149.153.23.52:2001 Location'
               }, 'Crypt::RSA::Key::Public' );

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 56803, ttl: 64)

TCP Source: 1253 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 40494
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

A Mobile Agent Registering/ACKing with the Key Server

With the RSA Public Key of the Key Server and the next Location known, the mobile agent (having generated a RSA Public/Private key-pairing of its own), needs to register its own RSA Public Key with the Key Server. The mobile agent begins by establishing a connection with the Key Server:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 46724, ttl: 64)

TCP Source: 1255 -> TCP Destination: 30002
TCP Header Length: 10, TCP Checksum: 1579
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 0, ttl: 63)

TCP Source: 30002 -> TCP Destination: 1255
TCP Header Length: 10, TCP Checksum: 4256
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 46725, ttl: 64)

TCP Source: 1255 -> TCP Destination: 30002
TCP Header Length: 8, TCP Checksum: 16181
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The mobile agent then sends its determined protocol port number (1254) to the Key Server, followed by its RSA Public Key:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 46726, ttl: 64)

TCP Source: 1255 -> TCP Destination: 30002
TCP Header Length: 8, TCP Checksum: 52929
TCP Data:

1254

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 49334, ttl: 63)

TCP Source: 30002 -> TCP Destination: 1255
TCP Header Length: 8, TCP Checksum: 16224
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 46727, ttl: 64)

TCP Source: 1255 -> TCP Destination: 30002
TCP Header Length: 8, TCP Checksum: 52812
TCP Data:

$VAR1 = bless( {
                 'e' => 65537,
                 'n' => '93193880704764181790541226256606738723205951026804 22
```

```
                        0011223467984748148249505728624808028427459955905929
                        8814728610345384128580078722025347843297530221484566
                        1727099972994261571041400994683889484947683266786058
                        7957649382984691694311968621882003354433362521008065
                        446975477797960409745932078769284945886728395447',
                'Version' => '1.91',
                'Identity' => 'Mobile::Executive ID10276exampleagent.pl'
              }, 'Crypt::RSA::Key::Public' );

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 46728, ttl: 64)

TCP Source: 1255 -> TCP Destination: 30002
TCP Header Length: 8, TCP Checksum: 15632
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The just-registered RSA Public Key is then requested from the Key Server:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 30465, ttl: 64)

TCP Source: 1256 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 5893
TCP Data:

149.153.23.51

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 19289, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1256
TCP Header Length: 8, TCP Checksum: 30581
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 30466, ttl: 64)

TCP Source: 1256 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 4307
TCP Data:

1254
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 19290, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1256
TCP Header Length: 8, TCP Checksum: 30577
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

As the registration has not yet occurred, the Key Server responds with the "`NOSIG`" token:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 19292, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1256
TCP Header Length: 8, TCP Checksum: 36929
TCP Data:
```

```
NOSIG
--end-sig--

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 19293, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1256
TCP Header Length: 8, TCP Checksum: 14102
TCP Data:

NOTFOUND
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 30468, ttl: 64)

TCP Source: 1256 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 30494
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The mobile agent *cannot* continue until the registration of its RSA Public Key is confirmed, so it sends another request to the Key Server:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 56865, ttl: 64)

TCP Source: 1257 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 57601
TCP Data:

149.153.23.51

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 10552, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1257
TCP Header Length: 8, TCP Checksum: 16754
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 56866, ttl: 64)

TCP Source: 1257 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 56015
TCP Data:

1254
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 10553, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1257
TCP Header Length: 8, TCP Checksum: 16750
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

This request is successful, and the Key Server responds with the appropriately formed and digitally signed RSA Public Key:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 10555, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1257
TCP Header Length: 8, TCP Checksum: 32681
TCP Data:

-----BEGIN RSA SIGNATURE-----
Version: 1.5
Scheme: Crypt::RSA::SS::PSS

0QAxMjgAU2lnbmF0dXJlgLhQC83CLlw320ZlXGLa3+c3tXseMNd6CRpDrKqasa9XWwETPd2amd3k
wVoUGfei8QI0tHM6h24GWDYVEqPFUP3dsWjZhhYIO++g1j+v0IvsGzMwwAo8kz0cTaIC/WKwZqwN
6fbr3565+FyNZ4POIdUBKDiiienX9aC8XCDVogo=
=CDui2jD/0QHb5K5hcxRnRA==
-----END RSA SIGNATURE-----

--end-sig--

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.51 (id: 10556, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1257
TCP Header Length: 8, TCP Checksum: 49264
TCP Data:

$VAR1 = bless( {
                 'e' => 65537,
                 'n' => '9319388070476418179054122625660673872320595102680422
                        0011223467984748148249505728624808028427459955905929
                        8814728610345384128580078722025347843297530221484566
                        1727099972994261571041400994683889484947683266786058
                        7957649382984691694311968621882003354433362521008065
                        446975477797960409745932078769284945886728395447',
                 'Version' => '1.91',
                 'Identity' => 'Mobile::Executive ID10276exampleagent.pl'
               }, 'Crypt::RSA::Key::Public' );

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.100.67 (id: 56868, ttl: 64)

TCP Source: 1257 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 15674
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The mobile agent can now relocate to the next Location, safe in the knowledge that its RSA Public Key is available from the Key Server.

A Mobile Agent Relocating from `pblinux` to `pbmac`

The Location running on `pbmac` (149.153.23.52, port 2001) begins by accepting a connection from the mobile agent running on protocol port number 1254 on `pblinux` (which has an IP

address of 149.153.23.51):

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.23.52 (id: 24184, ttl: 64)

TCP Source: 1254 -> TCP Destination: 2001
TCP Header Length: 10, TCP Checksum: 26723
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.23.51 (id: 0, ttl: 64)

TCP Source: 2001 -> TCP Destination: 1254
TCP Header Length: 10, TCP Checksum: 45332
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.23.52 (id: 24185, ttl: 64)

TCP Source: 1254 -> TCP Destination: 2001
TCP Header Length: 8, TCP Checksum: 57257
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The mobile agent sends the name of the program (exampleagent.pl) as well as the number of the next line to execute (12) to the Location:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.23.52 (id: 24186, ttl: 64)

TCP Source: 1254 -> TCP Destination: 2001
TCP Header Length: 8, TCP Checksum: 53897
TCP Data:

exampleagent.pl

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.23.51 (id: 54024, ttl: 64)

TCP Source: 2001 -> TCP Destination: 1254
TCP Header Length: 8, TCP Checksum: 57288
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.23.52 (id: 24187, ttl: 64)

TCP Source: 1254 -> TCP Destination: 2001
TCP Header Length: 8, TCP Checksum: 42075
TCP Data:

12

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.23.51 (id: 54025, ttl: 64)

TCP Source: 2001 -> TCP Destination: 1254
TCP Header Length: 8, TCP Checksum: 57285
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The mobile agent then sends the digitally signed source code to the Location:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.23.52 (id: 24188, ttl: 64)

TCP Source: 1254 -> TCP Destination: 2001
TCP Header Length: 8, TCP Checksum: 21027
TCP Data:

-----BEGIN RSA SIGNATURE-----
Scheme: Crypt::RSA::SS::PSS
Version: 1.5

0QAxMjgAU2lnbmF0dXJleu8Hh0aBb1cn0QT61C0GQUHRDjh0eb0JD4a3yp40XnQ2yMquqN3+2pTs
rjXthgXK2aKQg/YQCYK3TFKvzKCAaBazQBI1GbKjRpSecIK2osxgbvwx5K3BYEx/3pQkQW6CgGUs
V5gRdbNIsF0tTfSK0cpA77wBfjI+0m2XcQLQFG8=
=jVq0WfyQY7gukj/8/1sh2g==
-----END RSA SIGNATURE-----

--end-sig--

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.51 -> 149.153.23.52 (id: 24189, ttl: 64)

TCP Source: 1254 -> TCP Destination: 2001
TCP Header Length: 8, TCP Checksum: 17726
TCP Data:

-----BEGIN COMPRESSED RSA ENCRYPTED MESSAGE-----
Scheme: Crypt::RSA::ES::OAEP
Version: 1.24

eJwBEQLu/TEwADUxMgBDeXBoZXJ0ZXh0RP9hWpkXPADL/Y1doiQrnYrf8nkyw71jKcfnTRVcv0Q+
Doo21gYPfQf08N8s1tpXzHk3zlFjbK5kVVb6NCbCSN6XQGWUVp2sqxfGcxTFt6cjGym6qYzBDawC
GcpPY0x8PJk8C+/3w0Xy20X320Vxif37oLUmC1iDojP2SQoofrY1eY8KlSmQbkDY/K0u7tLX6L6N
FeirnS3upVEgHmEuJoADiY7ITs+Vlq9k1e3teP3hCNAe489Un6DSxrI5TnHQ9Q7TenD+xSlZz9XE
wh2hJ9Myj1Raf4n87duaobhP062yl7X5xmKBpMEZDRQF1zgzwcgcPmA8kp4I20TY3SwQcE3lKyZh
LyH+eu6kNP9W9ZD6vyDPBd9yi4PvltqfGWp76sN0LX2YZms9Yx6e00u5gQ5BjYn+ploweSBTiH6f
oZ4uRhnslKH56zqB4xia6aNTFG9oZsxIQoQZrnCHnFCv9IvZACFjH5JwSMkUa090rvDlubD+iNmy
6swEPSs7ZMtBYt9sRxAgmICFD4YSALvREsJfPgfTVIT/es0G0o8mTZ15oxczBtF3GrZ8sKUkf3B0
RlcIpouM68G/BebT/gWfiasqMhy00pyBjmfy0FQktGrEYhNZaB0W+oCpS4wEYNrqDEYANvVzdWEY
eUl/U/xaQcVqBNbCgyD/FrsZKp7fC0Jc1QTN
=CFlfFBxDuW8lY8is195YvQ==
-----END COMPRESSED RSA ENCRYPTED MESSAGE-----

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.23.51 (id: 54026, ttl: 64)

TCP Source: 2001 -> TCP Destination: 1254
TCP Header Length: 8, TCP Checksum: 56186
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

An Existing Location Requesting Keys from the Key Server

With the mobile agent now relocated to the Location, the next order of business is for the Location to request the RSA Public Key from the Key Server so that the digital signature can be verified prior to decryption. A connection is established with the Key Server running on `glasnost` (IP address 149.153.100.67):

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 24435, ttl: 64)

TCP Source: 1301 -> TCP Destination: 30001
TCP Header Length: 10, TCP Checksum: 52409
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.52 (id: 0, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1301
TCP Header Length: 10, TCP Checksum: 5541
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 24436, ttl: 64)

TCP Source: 1301 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 17466
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The Location then sends the IP address (149.153.23.51) and protocol port number (1254) of the RSA Public Key it is requesting to the Key Server:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 24437, ttl: 64)

TCP Source: 1301 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 58346
TCP Data:

149.153.23.51

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.52 (id: 33488, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1301
TCP Header Length: 8, TCP Checksum: 17499
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 24438, ttl: 64)

TCP Source: 1301 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 56760
TCP Data:

1254
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.52 (id: 33489, ttl: 63)
```

```
TCP Source: 30001 -> TCP Destination: 1301
TCP Header Length: 8, TCP Checksum: 17495
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

And the Key Server responds with the digitally signed RSA Public Key (which was recently registered by the mobile agent):

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 24439, ttl: 64)

TCP Source: 1301 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 17446
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.52 (id: 33490, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1301
TCP Header Length: 8, TCP Checksum: 17489
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.52 (id: 33491, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1301
TCP Header Length: 8, TCP Checksum: 5682
TCP Data:

-----BEGIN RSA SIGNATURE-----
Version: 1.5
Scheme: Crypt::RSA::SS::PSS

OQAxMjgAU2lnbmF0dXJlJ6KtRGRrz/KT7JUxxCYSNsQiDvyeUsrcJqU30JfUsb7n0cNBVGCtKTIs
u5+n5kvC3jkbhtBTqocvtriNh8gM/KmBR4I7AaFm+Fwv4NUwIhQk4UUU9xEbZ0xGPzc7rTmv+1lY
c4+K0AruYkGGfQdYgnAK7Yb3InrFpFtcsqg3Irg=
=aKmEQ75ORo78rioDsnwnxQ==
-----END RSA SIGNATURE-----

--end-sig--

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.23.52 -> 149.153.100.67 (id: 24440, ttl: 64)

TCP Source: 1301 -> TCP Destination: 30001
TCP Header Length: 8, TCP Checksum: 16420
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
149.153.100.67 -> 149.153.23.52 (id: 33492, ttl: 63)

TCP Source: 30001 -> TCP Destination: 1301
TCP Header Length: 8, TCP Checksum: 50010
TCP Data:

$VAR1 = bless( {
                  'e' => 65537,
                  'n' => '93193880704764181790541226256606738723205951026804220
                          0011223467984748148249505728624808028427459955905929
                          8814728610345384128580078722025347843297530221484566
                          1727099972994261571041400994683889484947683266786058
```

```
                              79576493829846916943119686218820033544333625210080065
                              44697547779796040974593207876928494588672839544 7',
                      'Version' => '1.91',
                      'Identity' => 'Mobile::Executive ID10276exampleagent.pl'
                    }, 'Crypt::RSA::Key::Public' );

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    149.153.23.52 -> 149.153.100.67 (id: 24441, ttl: 64)

    TCP Source: 1301 -> TCP Destination: 30001
    TCP Header Length: 8, TCP Checksum: 14631
    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

As a result of all of this activity, a number of additional screen status messages were generated by the Key Server, as follows:

```
Checking the PK+ value for 127.0.0.1/30001.
  -> No need to sign PK+ for keyserver.
  --> Sending SELFSIG to client (149.153.23.51).
  ---> Sending PK+ for 127.0.0.1/30001 to client.
Checking the PK+ value for 149.153.23.52/2001.
  -> Signing PK+ for 149.153.23.52/2001.
  --> Sending signature to client (149.153.23.51).
  ---> Sending PK+ for 149.153.23.52/2001 to client.
Checking the PK+ value for 149.153.23.51/1254.
  -> Sending NOSIG to client (149.153.23.51).
  --> Sending NOTFOUND for 149.153.23.51/1254 to client.
[INSERT] Inserting the 149.153.23.51/1254 pairing.
Checking the PK+ value for 149.153.23.51/1254.
  -> Signing PK+ for 149.153.23.51/1254.
  --> Sending signature to client (149.153.23.51).
  ---> Sending PK+ for 149.153.23.51/1254 to client.
Checking the PK+ value for 149.153.23.51/1254.
  -> Signing PK+ for 149.153.23.51/1254.
  --> Sending signature to client (149.153.23.52).
  ---> Sending PK+ for 149.153.23.51/1254 to client.
```

Note the initial failure to lookup the RSA Public Key prior to its successful insertion. The `SCOOBY.publics` database table is also updated as a result of all this activity. A new entry has been added for the mobile agent running on protocol port number 1254 on `pblinux` (IP address 149.153.32.51):

```
mysql> select ip_address, protocol_port from publics;
+---------------+---------------+
| ip_address    | protocol_port |
+---------------+---------------+
| 127.0.0.1     | 30001         |
| 149.153.23.52 | 2001          |
| 149.153.23.51 | 1254          |
+---------------+---------------+
3 rows in set (0.00 sec)
```

A query on the database table confirms that the RSA Public Key within the Key Server is as was requested:

```
mysql> select public_key from publics where ip_address = '149.153.23.51';
+----------------------------------------------------------------------------+
| public_key                                                                 |
+----------------------------------------------------------------------------+
| $VAR1 = bless( {                                                           |
|                  'e' => 65537,                                             |
|                  'n' => '93193880704764181790541226256606738723205951026804222 |
|                          00112234679847481482495057286248080284274599559059929 |
|                          88147286103453841285800787220253478432975302214845666 |
|                          17270999729942615710414009946838894849476832667860588 |
|                          79576493829846916943119686218820033544333625210080655 |
|                          446975477797960409745932078769284945886728395447',  |
|                  'Version' => '1.91',                                      |
|                  'Identity' => 'Mobile::Executive ID10276exampleagent.pl'  |
|                }, 'Crypt::RSA::Key::Public' );                              |
+----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Chapter 6

Analysis

This chapter presents a critical analysis of the delivered facility. It discusses the original design goals of the facility, indicating those goals that have been met and detailing those that have not. A list of potential, future enhancements to each of the components of the facility concludes this chapter.

6.1 Satisfying the Overall Goal

This project proposed the development of a set of extensions to the Perl programming language that would allow any Perl programmer to write mobile agents. Specifically, the `Mobile::Executive` and `Mobile::Location` modules would provide a facility whereby a programmer could code mobile agents (clients) and Locations (servers), respectively. A *hybrid API* was proposed, in that the developed facility would support programmers from the procedural and object-oriented worlds.

In the broadest sense, the delivered facility meets these proposed goals. The two modules exist. The `Mobile::Executive` module provides a procedural API to the mobile agent programmer, whereas the `Mobile::Location` class provides an object-oriented API to Location developers.

6.2 Satisfying the Requirements

This section briefly discusses the various increments from the Requirements chapter of this document for both the Location and mobile agent technologies, indicating those requirements that have been met, and those that have not.

The Location Increments

The four Location increments were:

1. An Unsafe Location - the Location executes received code without a care in the world.
2. A Safe Location - the Location executes received code within a restricted compartment.
3. An Authenticated Location - the Location only executes received code that it can prove to be authentic.
4. An Embedded Location - the Location operates within a web-server.

With each iteration of the Location technology, each of these increments were developed. Unfortunately, the development and integration of the third increment resulted in an incompatibility with the second. This resulted is a hard choice: allow only a safe Location where any received code executes within a restricted compartment, or allow only an Authenticated Location where any received code is assumed to be trustworthy in that it has been received from a trusted source. The code to support the safe Location was developed and did work, but had to be disabled whenever the Authenticated code was developed and integrated into the Location technology[1]. As the latter technology proved to be more interesting and significantly more of a challenge to get working, it was favored over the safe Location technology. It is proposed to return to this issue at a later date when the incompatibility is addressed by the developers of the third-party `Crypt::RSA` module (which caused the problem).

[1] Refer to the document entitled *Annotations to the Source Code* for details on this, especially the annotations to the `_service_client` method. This document is included on the accompanying CD.

As for the fourth increment of the Location Technology, the decision to embed a Location within a web server was reversed. Upon reflection, it seemed a much better idea to embed a web server within the Location in order to provide a remote monitoring facility accessible from any web browser. Consequently, the fourth increment morphed to become the addition of the Web-based Monitoring Service within each Location.

The Mobile Agent Increments

Five mobile agent increments were specified, as follows:

1. Support for the relocation of simple statements (code only, no state).
2. Additional support for the relocation of simple statements and simple variables (code and state).
3. Additional support for the relocation of objects.
4. Additional support for the relocation of open disk-files.
5. Additional support for the relocation of any required Perl modules (not currently available on the next Location).

Of all of these increments, only the fourth was not attempted (although it was considered). Increment 1, 2 and 3 are implemented within the facility and 5 was attempted but had to be abandoned due to module compatibility problems.

The accompanying CD contains a document called *Example Mobile Agents* which presents the source code to a selection of mobile agents developed to explicitly exercise the first, second and third mobile agent increments. These programs are based heavily on those specified within the Requirements chapter of this document.

Problems Relocating Open Disk-Files

Why not even attempt the fourth mobile agent increment? Surely if a mobile agent is using a open disk-file on one Location, then decides to relocate to another Location, the disk-file

should travel too, so as to allow the contents of the disk-file to be available whenever the mobile agent re-executes?

Asking such a question is reasonable. However, consider the disk-file being processed on the first location is the Linux password file (`/etc/passwd`). Where on the next Location should it reside after relocation? Place it in `/etc/passwd` and the next Location's password file has just been overwritten, an action that is highly unlikely to be the required behaviour. Place it relative to the directory that is executing the mobile agent (in `etc/passwd`), and the mobile agent may fail as its expecting the file to be in the same place that it was in prior to relocation. What to do? The *safest* thing to do is to do nothing. So, mobile agent increment four was not implemented, as it is better to be safe than sorry.

Further, the ability to relocate open disk-files could not be implemented in an "automatic" way. Consider the relocation of simple variables. By declaring the variables as "my" variables[2], the facility can automatically determine the list of lexicals and relocate them. Other than using the "my" keyword to declare the variable (which is an established practice within the Perl programming community), the programmer need do nothing else to signal a desire to relocate the variable. This provides a very natural API. The intent was to provide a similar "automatic, natural" API for disk-file relocation: the programmer would simple open a disk-file, use it, relocate, and have the file appear "automatically" on the next Location. This proved to be quite a technical challenge, that - ultimately - resulted in the increment being abandoned. Consideration was given to the possibility of somehow marking the open disk-file as relocatable, perhaps using code something like this:

```
open FH, "test.txt" :relocatable;

$line = <FH>;   # Read a line from file.

relocate( localhost, 3333);

$line = <FH>    # Read another line from file.
```

Which places the burden of deciding which disk-files to relocate very much on the programmer. Although similar to the "marking" of variables with "my", it is not as natural.

[2]Which tells the Perl interpreter to treat the variable as lexically scoped to its block, as opposed to scoped to the entire source code file.

It is possible to assign a open disk-file to a “my” variable. However, when examined by the mechanism that relocates simple variables, the `Scooby.pm` module was unable to distinguish between a “my” variable that referred to an open disk-file and one that referred to a simple variable. So, it was not possible to leverage the existing mechanism to relocate open disk-files.

Another possibility is to support relocation of open disk-files that are contained in the same directory as the mobile agent (which would include disk-files within subdirectories off the mobile agent's directory). The reasoning here is that disk-files in these directories are somehow related to the mobile agent, so relocating them to a similar directory structure on the next Location is an assumption that might hold. Care would be needed, though. Specifically, it would be necessary to determine the starting directory of the mobile agent, not its current working directory. Consider a mobile agent that begins by changing directory to `/etc/`, then opening the `passwd` disk-file. If relocation is then requested, the `passwd` disk-file is relocated to the next Location with disastrous results, as the disk-file is within the mobile agent's current working directory. Consequently, the facility would have to ensure that it was able to determine that relocation of open disk-files from directories other that the one within which the mobile agent started is invalid. Additionally, the facility needs to ensure that a deeply nested disk-file once relocated, appears within a similarly nested directory tree on the next Location. *It all gets very complicated, very quickly.* Unlike the relocation of variables, where we can make reasonable assumptions about the mobile agent actually owning the data, the same cannot be said for open disk-files.

Problems Relocating Used Modules

The fifth mobile agent increment hoped to leverage the `Class::Tom` module, which is put to great use within Steve Purkis' *Agent Perl*. Regrettably, the `Class::Tom` module has not seen any development for a number of years, and does not work with the most recent versions of Perl. Time did not permit a wholesale exploration of what would be required to amend `Class::Tom` to work within the facility. Consequently, increment 5 was not implemented.

However, following the public release of *Scooby*, the developer responsible for `Class::Tom` may well fix the problem.

6.3 Linux Only Implementation (For Now)

It had been hoped to develop the facility to support at least two computing platforms, specifically the Linux operating system and the Windows platform. Although Perl as a programming technology is very portable (in that the current release of Perl supports a staggering number of platforms, see [Perl5Porters, 2003f]), the availability of certain third-party modules on all platforms can be somewhat “hit-or-miss”. For instance, the `PadWalker` module (which is central to the correct functioning of the facility) has yet to be ported to the Windows platform. Consequently, it became necessary to limit this initial release of the facility to Linux and Unix platforms only. As Perl modules become more popular, they tend to be ported to other platforms over time. As the `PadWalker` module used in this project is at version 0.08, it is anticipated that it will see many more releases in the near future and that, as it popularity grows, it will be ported to more platforms[3].

6.4 Satisfying the Design

Four mechanisms were identified within the Design Phase as being deliverables: the Relocation, Security, Transport and Load-and-Execute mechanisms. Although not all of the design goals were met, a version of all four mechanisms is delivered within this release of the facility.

6.4.1 The Delivered Relocation Mechanism

The Relocation Mechanism is based on the *Modify the Mobile Agent Source Code* technique described in the Design chapter. Although somewhat crude, this technique proved to be more than adequate. The alternative, preferred technique, *Transmit Perl Bytecode*, proved

[3]On May 21, 2003, the `PadWalker` maintainer released version 0.09 of the module. This release appeared too late to be integrated and tested with the facility.

to be so difficult that it was abandoned in favour of the working, prototype implementation. In a time-honored tradition among many programmers, it was a case of "if it is not broken, do not fix it".

6.4.2 The Delivered Security Mechanism

The Security Mechanism (or mechanisms, to be more correct) was always going to be complex. The Design document explored a number of techniques for securing mobile agents in transit, protecting against malicious Locations, and guarding against malicious mobile agent programs. Upon reflection, and after further research, it became clear that some of the proposed security techniques were somewhat naive. A security system based on public key cryptography, described in [Stallings, 2000b], and a centralized Key Server provide an environment within which clients *and* servers authenticate immediately prior to *and* after relocation. Encryption and decryption, in addition to the use of digital signatures, protect mobile agents in transit in addition to the Locations. The security system is based on trust, not on technology. By relying on a public-key cryptosystem, the requirement to employ traditional, symmetric cryptosystems and the Diffie-Hellman Key Exchange Protocol were negated. The delivery of the Key Server enables the secure dispersal of public keys within the facility. Note that the provision of the Key Server was not envisaged as part of the initial design of this project.

6.4.3 The Delivered Transport and Load-and-Execute Mechanisms

The Transport Mechanism is implemented as described in the Design chapter with the notable exception that there is no facility to communicate over UDP. To transport data securely presupposes inherent reliability, which is something that UDP makes no attempt to deliver. The Load-and-Execute Mechanism is implemented with TCP as described in the Design chapter.

6.5 Critical Assessment

This section discusses the project as a whole and offers a critical assessment of the delivered facility.

6.5.1 How Strong is Scooby?

Based on the mobile agent classification first presented in [Fuggetta, 1998] (and repeated in [Tanenbaum, 2001]), the facility can be classified as supporting *sender-initiated process migration*. That is, the facility supports *strong mobility*, in that not only the *code* by also the *state* of the mobile agent is transported, effectively enabling the resumption of execution from where it left off. In this regard, the project has been a success. However, there are some restrictions placed on the mobile agent writer by the facility that may limit its appeal to some programmers[4]. If a programmer can live with the restrictions, capable mobile agents can be developed quickly and easily, then deployed within a secure environment.

6.5.2 How Strong is the Key Server?

The strength of the facility (from a security perspective) depends heavily on the level of security achieved by the Key Server. If the Key Server is compromised (or attacked), the entire facility suffers. If the Key Server crashes, all relocation is halted. As such, the Key Server is a single-point-of-failure, which is never a good idea within a distributed system. As the community that the Key Server serves gets larger, the central role that the Key Server plays becomes more of an issue. Centralized solutions (such as that employed by the facility) tend to scale poorly. Another problem is that of *impersonation*[5]. What happens if the Key Server is attacked by a denial-of-service attack, crashes, then is replaced by another malicious Key Server impersonating the first one? Such attacks are notoriously difficult to protect against, let alone detect.

A number of techniques can help with some of the issues surrounding the current

[4]For details, refer to *The Scooby Guide* which is included on the accompanying CD.
[5]Also known as "spoofing".

implementation of the Key Server. These include the use of *certificates* to authenticate public keys (effectively removing the need to run the Key Server 24 x 7), implementing client-side caching (and reuse) of public keys, and introducing redundancy to the Key Server, by replicating the database over a number of machines. This, of course, introduces its own problems, as correctly replicating databases within a distributed environment is not easy.

The delivered Key Server provides the technology to support the RSA cryptosystem built into the facility. Is it not perfect, but it does work.

6.5.3 What Actually Happens: An Example

Consider the following mobile agent, called `drivespace`, which relocates to a series of Locations and determines the amount of disk-space used at each Location (by simply running the `df` command-line utility and capturing its output):

```
#! /usr/bin/perl -w

#
# drivespace
#

use strict;
use Mobile::Executive;

my %space = ();

sub do_it {
    # This code is executed at each Location.

    return scalar `df -Th`;
}

relocate( 'pbmac.itcarlow.ie', 2001 );
$space{ 'pbmac.itcarlow.ie' } = do_it;

relocate( 'pblinux.itcarlow.ie', 4001 );
$space{ 'pblinux.itcarlow.ie' } = do_it;

relocate( 'testimac.itcarlow.ie', 3001 );
$space{ 'testimac.itcarlow.ie' } = do_it;

relocate( 'pblinux.itcarlow.ie', 4001 );

foreach my $host ( keys %space )
{
    print "$host reported diskspace of: \n\n$space{ $host }\n";
}
```

The code is clear, well-written and structured. Now, take a look at what this code mutates into as a result of the four relocations:

```
#! /usr/bin/perl -w
goto LABEL_drivespace631053950179;
 goto LABEL_drivespace431053953691;
  goto LABEL_drivespace291053950085;
   goto LABEL_drivespace191053953737;

    #
    # drivespace
    #

    use strict;
    use Mobile::Executive;

    my %space = ();

    sub do_it {
        # This code is executed at each Location.

        return scalar `df -Th`;
    }

    relocate( 'pbmac.itcarlow.ie', 2001 );
   LABEL_drivespace191053953737:
   1;
   use Mobile::Executive;

        %space = (
        );
    $space{ 'pbmac.itcarlow.ie' } = do_it;

    relocate( 'pblinux.itcarlow.ie', 4001 );
  LABEL_drivespace291053950085:
  1;
  use Mobile::Executive;

       %space = (
           "pbmac.itcarlow.ie" =>
   "Filesystem    Type    Size  Used Avail Use% Mounted on
   /dev/sda10    ext2    3.1G  2.3G  654M  79% /
   none         tmpfs     61M     0   60M   0% /dev/shm
   ",
       );
    $space{ 'pblinux.itcarlow.ie' } = do_it;

    relocate( 'testimac.itcarlow.ie', 3001 );
 LABEL_drivespace431053953691:
 1;
 use Mobile::Executive;

      %space = (
          "pbmac.itcarlow.ie" =>
   "Filesystem    Type    Size  Used Avail Use% Mounted on
   /dev/sda10    ext2    3.1G  2.3G  654M  79% /
```

```
      none          tmpfs      61M      0   60M   0% /dev/shm
      ",
             "pblinux.itcarlow.ie" =>
     "Filesystem    Type    Size  Used Avail Use% Mounted on
     /dev/hda3     ext3    4.3G  3.0G  1.1G  71% /
     /dev/hda2     ext3     99M   38M   55M  41% /boot
     none         tmpfs     30M     0   30M   0% /dev/shm
     /dev/hda1     vfat    4.7G  2.8G  1.9G  59% /mnt/win
     ",
        );
      $space{ 'testimac.itcarlow.ie' } = do_it;

      relocate( 'pblinux.itcarlow.ie', 4001 );
    LABEL_drivespace631053950179:
    1;
    use Mobile::Executive;

       %space = (
            "pbmac.itcarlow.ie" =>
      "Filesystem    Type    Size  Used Avail Use% Mounted on
      /dev/sda10    ext2    3.1G  2.3G  654M  79% /
      none          tmpfs      61M      0   60M   0% /dev/shm
      ",
            "pblinux.itcarlow.ie" =>
     "Filesystem    Type    Size  Used Avail Use% Mounted on
     /dev/hda3     ext3    4.3G  3.0G  1.1G  71% /
     /dev/hda2     ext3     99M   38M   55M  41% /boot
     none         tmpfs     30M     0   30M   0% /dev/shm
     /dev/hda1     vfat    4.7G  2.8G  1.9G  59% /mnt/win
     ",
            "testimac.itcarlow.ie" =>
    "Filesystem    Type    Size  Used Avail Use% Mounted on
    /dev/hda9     ext2    2.8G  2.0G  698M  75% /
    ",
       );

      foreach my $host ( keys %space )
      {
          print "$host reported diskspace of: \n\n$space{ $host }\n";
      }
```

Which, admittedly, looks like a bit of a mess. This is especially true of the four `goto` statements at the top of the source code, as they are guaranteed to attract scorn from every computer scientist that reads them. Yes, this code is awful: it is no longer clear, well-written nor reasonably structured. However, this does not matter. Once this program starts to relocate, it is mutated and executed by the `Scooby.pm` module and the `Location.pm` class. The mutations are never meant to be read (nor amended) by a human reader. If the behaviour of the mobile agent needs to be adjusted, changes are made to the original file,

not the mess that it mutates into. So, using `goto` in this way, and mutating source code in this way is an explainable practice. After all, it is the results of the execution of the mobile agent that are of interest, and they are displayed on the screen of the last relocated-to Location, as follows:

```
pblinux.itcarlow.ie reported diskspace of:

Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/hda3     ext3    4.3G  3.0G  1.1G  71% /
/dev/hda2     ext3     99M   38M   55M  41% /boot
none         tmpfs     30M     0   30M   0% /dev/shm
/dev/hda1     vfat    4.7G  2.8G  1.9G  59% /mnt/win

pbmac.itcarlow.ie reported diskspace of:

Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/sda10    ext2    3.1G  2.3G  654M  79% /
 none         tmpfs     61M     0   60M   0% /dev/shm

testimac.itcarlow.ie reported diskspace of:

Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/hda9     ext2    2.8G  2.0G  698M  75% /
```

6.6 Future Enhancements

Half the battle with any project is knowing when to stop. This project "stopped" when the vast majority of its design goals had been achieved. However, there's always more to do, so this section offers a number of lists of suggested future enhancements to the facility's component parts.

6.6.1 Enhancing the Key Server

1. Amend the Key Server to query the `SCOOBY.publics` table with all of the data fields, that is: `ip_address`, `protocol_port` and `public_key`. This avoids the potentially disastrous, but subtle, scenario discussed earlier in this document.

2. Allow the Web-based Monitoring Service to run on a programmer-defined protocol port number, not just 8080.

3. Provide a password-protection mechanism to the Web-based Monitoring Service. At the moment, *anyone* with a browser can access the service and reset the log-file. Investigate possibility of using SSL.

4. Investigate the possibility that working with the Log-file may result in a "race condition".

5. Convert all the procedural Socket API code to use the object-oriented API provided by the `IO::Socket` API.

6.6.2 Enhancing `Scooby.pm`

1. Provide for better error checking.

2. Investigate possibility of extracting the Web-based Monitoring Service support code from this module, creating another module, then sharing the code with `Location.pm`.

3. Support the relocation of nested Perl data structures (for example, arrays of arrays, hashes of hashes, arrays of hashes, etc., etc.).

4. Investigate the possibility of signing/encrypting the line number and filename communications between the mobile agent and the next Location. At the moment, this communication occurs as "clear text", and would therefore be susceptible to attack and corruption.

5. Investigate the possibility of performing all of the mutation on one side of the communication. At the moment, both the `Scooby.pm` and `Location.pm` components perform mutations on the source code. It should be possible to mutate only once.

6.6.3 Enhancing `Location.pm`

1. Allow the Web-based Monitoring Service to run on a programmer-defined protocol port number, not just 8080.

2. Provide a password-protection mechanism to the Web-based Monitoring Service. At the moment, *anyone* with a browser can access the service and reset the log-file. Investigate possibility of using SSL.

3. Investigate the possibility that working with the Log-file may result in a "race condition".

4. Provide for better error checking.

6.6.4 Enhancing `Executive.pm`

1. Provide for better error checking.

Chapter 7

Conclusions

This chapter begins by offering some personal comments on the project, then discusses some conclusions drawn from the work.

7.1 Some Personal Comments

With the project complete, it is possible to reflect back on the process and offer some personal comments.

Over the lifetime of this project, I completely immersed myself in Perl, its language, culture and community. This allowed me to produce what I believe to be a facility that Perl programmers will actually want to use and enjoy. I also benefited from the recent publication of a number of books, including [Tregar, 2002], [Jenness, 2003] and [Wall, 2000]. These proved to be essential references to understanding the inner workings of Perl. [Conway, 2000] and [Stein, 2001] also proved highly useful texts.

The Perl language went through two major overhauls during the time that this project was "active", culminating in the availability of release 5.6.1 and 5.8.0 of the language. It is annoying that the facility is incompatible with the 5.8.0 release of the language (the most recent) due to a problem with the `Crypt::RSA` module, which provides RSA support to the facility, and which does not currently work properly with the 5.8.0 release.

But for these Perl language release difficulties, I had no problems with Perl as the

implementation language. It is not a perfect programming language by any means but, I cannot imagine delivering the equivalent functionality with another language. Perl is my programming language of choice, and will remain so for some time.

By far the most interesting and challenging aspect of the project was discovering and then developing the Relocation Mechanism. The realization that what was needed was a debugger kicked-started the coding/testing phase, which ran from October 2002 through to early May 2003. By employing the *Incremental Model*, it was possible to quickly produce a functioning Location that could receive a simple mobile agent. These simple technologies were then enhanced (and iterated over) to eventually produce the delivered facility.

Producing a system that operated securing using the RSA technologies was also very interesting. This is the first large scale project that I have been involved in that uses a cryptosystem, and it was often a challenge to get the various crypto-processes right "in my head" before committing them to source code.

There were disappointments and dead-ends along the way, but this is to be expected when working on a project of this size. Overall, I believe that working on this project has been worthwhile and I hope that the delivered facility supports this conclusion.

On June 4th, 2003, the facility was uploaded to CPAN, the global Perl archive on the Internet, and is available at `http://search.cpan.org/author/BARRYPJ/`. With this act comes a fair amount of responsibility, ongoing commitment and maintenance. It is my hope that the small number of Perl programmers using Perl for mobile agent programming will consider my technology for their next project.

7.2 Unanswered Questions

Two unanswered questions remain:

1. Despite being of interest from a purely technical point-of-view, are mobile agents actually useful?

2. Are mobile agents destined to be a pervasive technology?

Answers to these questions are offered in the sections which follow.

7.2.1 Are Mobile Agents Useful?

Most (if not all) mobile agents can be developed as client/server applications, and vice-versa. However, mobile agents suggest a new server paradigm: *truly dynamic Internet services*. To explain, consider that typical client/server Internet systems provide a fixed set of services. A server allows a client to request a service to be performed, and then responds appropriately. As long as the client lives by the rules, by adhering to the defined protocol, all is well. However, the client/server model cannot support any real form of dynamic execution. Note that "dynamic" web-page creation technologies, despite their name, are still fixed services. The content is dynamic, but the underlying service is fixed.

By providing a general mechanism within which a client can dictate the service to be performed by the server (in the form of a mobile agent), the paradigm shifts from server-controlled to client-controlled. Thus, truly dynamic Internet services are possible. And truly dynamic Internet services are useful. However, do "possible" and "useful" ensure mobile agents will happen? Are mobile agents the next-big-thing? Will they be pervasive?

7.2.2 Will Mobile Agents be Pervasive?

Despite the support of the mobile agent communities (see [Kotz, 1999] and [Lange, 1999]), it seems unlikely that mobile agent technologies will ever truly catch on in a general sense. They are useful, and they can result in some elegant solutions to problems, but the Internet within which mobile agents now find themselves is a very different place toward the Internet that existed when mobile agents initially generated a lot of research interest (the middle to late 1990's). As the Internet has grown, so has the security concerns of those that connect to it. And in the end, security is all about control. The current Internet technologies are server-based, and centrally controlled, which makes controlled security possible. By fixing the set of services available to clients, server administrators can protect against attacks on those services. The mobile agent paradigm wrenches this control away from the server

administrator, and hands it to the client. The client (that is, the mobile agent), now has free reign. Although it can be shown that technologies exist to guard against inappropriate mobile agent actions, the very thought of remote, “foreign” code executing freely on a server is enough to turn the vast majority of the Internet’s systems administrators against the idea. There is nothing inherently wrong with mobile agent technology. It is the perceived security threat that may ultimately kill off the very idea of mobile agents as a pervasive technology.

Bibliography

[Aas, 2003] The `HTTP::Daemon`, available on CPAN as part of the `libwww-perl` library, see [CPAN, 2003]. Written by Gisle Aas.

[Ajanta, 2003] `http://www.cs.umn.edu/Ajanta`, *The Ajanta System*, a Java-based Mobile Agent System.

[Barry, 2002] Barry, P., *Programming the Network with Perl*, John Wiley & Sons Limited, ISBN: 0-471-48670-1, 2002.

[Chapman, 1997] Chapman, N., *Perl: The Programmer's Companion*, John Wiley & Sons Ltd., ISBN: 0-471-97563-X, 1997.

[Chess, 1995] Chess, D., Harrison, C. and Kershenbaum, A., *Mobile Agents: Are They A Good Idea?*, IBM Research Report, Almaden, T. J. Watson Research Center, Yorktown Heights, 1994/1995.

[Christiansen, 1998] Christiansen, T. and Torkington, N., *Perl Cookbook*, O'Reilly and Associates Inc., ISBN: 1-56592-243-3, 1998.

[Cockayne, 1998] Cockayne, W. T. and Zyda, M., *Mobile Agents*, Manning Publications Co., (Prentice-Hall, Professional Technical Reference), ISBN: 1-884777-36-8, 1998.

[Comer, 1999] Comer, D., *Computer Networks and Internets, Second Edition*, Prentice-Hall, 1999, pages 325-326.

[Conway, 2000] Conway, D., *Object Oriented Perl*, Manning Publications Co., ISBN: 1-884777-79-1, 2000.

[CPAN, 2003] Various authors, *CPAN, the Comprehensive Perl Archive Network*, available on-line at: `http://www.cpan.org`.

[Diffee, 1976] Diffie, W. and Hellman, M., *New Directions in Cryptography*, IEEE Transactions in Information Theory, volume IT-22, no. 6, pages 644-654, 1976.

[Dijkstra, 1968] Dijkstra, E. W., *Go To Statement Considered Harmful*, Vol. 11, No. 3, March 1968, pp. 147-148, Association for Computing Machinery (Communications of the ACM), 1968.

[Dubios, 1999] Dubois, P., *MySQL*, New Riders, ISBN: 0-737-709211, 1999.

[Franklin, 1996] Franklin, S. and Graesser, A.: *Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents*, Institute for Intelligent Systems, University of Mtextit, from the Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996. This article is available on-line at: `http://www.msci.memphis.edu/~franklin/AgentProg.html`.

[Fuggetta, 1998] Fuggetta, A., Picco, G. P. and Vigna, G., *Understanding Code Mobility*, IEEE Transactions on Software Engineering, volume 24, no. 5, pages 342-361, May 1998.

[Garfinkel, 1995] Garfinkel, S., *PGP: Pretty Good Privacy*, O'Reilly & Associates, 1995.

[Hylton, 1996] Hylton, J., Manheimer, K., Drake Jr., F. L., Warsaw, B., Masse, R. and van Rossum, G., *Knowbot programming: System support for mobile agents*, Proceedings of the Fifth International Workshop on Object Orientation in Operating Systems (IWOOOS'96), Seatle, Wash., 1996.

[Hylton, 1997] Hylton, J., and van Rossum, G., *Using the Knowbot Operating System in a Wide-Area Network*, 3rd ECOOP Workshop on Mobile Object Systems, Jyvaskyla, Finland, June 1997.

[IANA, 2003] The Internet Assigned Numbers Authority, on-line at: `http://www.iana.org`.

[IETF, 1974] Internet Engineering Task Force, *Transmission Control Protocol* as documented in *RFC 793*, available on-line at `http://www.ietf.org`.

[Jenness, 2003] Jenness, T. and Cozens, S., *Extending and Embedding Perl*, Manning Publications Co., ISBN: 1-930110-82-0, 2003.

[Kotz, 1999] Kotz, D. and Gray, R. S., *Mobile Agents and the Future of the Internet*, Department of Computer Science, Dartmouth College, Hanover, New Hampshire, 1999. Available on-line at: `www.cs.dartmouth.edu/~dfk/papers/kotz:future2/`.

[Lange, 1998] Lange, D. B. and Oshima, M., *Programming and deploying Java Mobile Agents with Aglets*, Addison-Wesley Longman, 1998.

[Lange, 1999] Lange, D. B. and Oshima, M., *Seven good reasons for mobile agents*, Communications of the ACM, 42(3): 88-89, March 1999.

[Meltzer, 2001] Meltzer, K. and Michalski, B., *Writing CGI Applications with Perl*, Addison-Wesley, ISBN: 0-201-71014-5, 2001.

[Openssh, 2003] `http://www.openssh.org`, *The OpenSSH Project*, an open-source implementation of the SSH Suite of Protocols.

[O'Reilly, 1999] O'Reilly, T., *Hardware, Software, and Infoware*, published essay as part of *Open Sources: Voices from the Software Revolution*, O'Reilly and Associates Inc., ISBN: 1-56592-582-3, 1999. Available on-line at: `http://www.oreilly.com/catalog/opensources/book/toc.html`.

[Perl5Porters, 2003a] The Perl 5 Porters, *Perl Debugger, the "perldebug" manual page*, part of the standard on-line documentation to Perl[Perl5Porters, 2003e].

[Perl5Porters, 2003b] The Perl 5 Porters, *Guts of Perl Debugging, the "perldebguts" manual page*, part of the standard on-line documentation to Perl[Perl5Porters, 2003e].

[Perl5Porters, 2003c] The Perl 5 Porters, *The Perl Manual Pages*, available as part of the Perl binary and source distributions, see [Perl5Porters, 2003e].

[Perl5Porters, 2003d] The Perl 5 Porters, *Perl Modules: How They Work, the "perlmod" manual page*, part of the standard on-line documentation to Perl[Perl5Porters, 2003e].

[Perl5Porters, 2003e] The Perl 5 Porters, `http://www.perl.com/CPAN/src/stable.tar.gz` links to the source code (which includes the on-line documentation) for the Perl programming language and environment.

[Perl5Porters, 2003f] The Perl 5 Porters, *List of Perl Supported Platforms*, available on-line at: `http://www.perl.com/CPAN/ports/`.

[Pressman, 1997] Pressman, R. S., *Software Engineering: A Practitioner's Approach, Fourth Edition*, McGraw-Hill, ISBN: 0-07-052182-4, 1997.

[Purkis, 1997] Purkis, S., *Agent Perl Website*, available on-line at: `http://tiamat.epn.nu/~spurkis/Agent/`.

[RSA, 2003] `http://www.rsasecurity.com`, The RSA Web-site.

[Scott, 2001] Scott, P. and Wright, E., *Perl Debugged*, Addison-Wesley, ISBN: 0-201-70054-9, 2001.

[Singh, 1999] Singh, S., *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, Fourth Estate Ltd., ISBN: 1-85702-879-1, 1999.

[Stallings, 2000a] Stallings, W., *Data and Computer Communications, Sixth Edition*, Prentice Hall, 2000.

[Stallings, 2000b] Stallings, W., *Network Security Essentials: Applications and Standards*, Prentice-Hall, ISBN: 0-13-016-93-8, 2000.

[Stein, 2001] Stein, L. D., *Networking Programming with Perl*, Addison-Wesley, ISBN: 0-201-61571-1, 2001.

[Stevens, 1998] Stevens, R., *UNIX Networking Programming: Networking APIs: Sockets and XTI*, Prentice Hall, ISBN: 0-13-490012-X, 1998.

[Sundsted, 1998] Sundsted, T., *An Introduction to Agents*, JavaWorld, Web Publishing Inc., (IDG), June 1998. Available on-line at: `http://www.javaworld.com/jw-06-1998/jw-06-howto.html`.

[Tanenbaum, 2001] Tanenbaum, A. and van Steen, M., *Distributed Systems: Principles and Paradigms*, Prentice-Hall Inc., ISBN: 0-13-088893-1, 2001.

[TCL, 2003] `http://tcl.activestate.com`, *The Tcl Developer Site*, maintained by ActiveState Corporation.

[Tregar, 2002] Tregar, S., *Writing Perl Modules for CPAN*, A!Press, ISBN: 1-590-59018X, 2002.

[Wall, 2000] Wall, L., Christiansen, T. and Orwant, J., *Programming Perl, Third Edition*, O'Reilly & Associates, Inc., ISBN: 0-596-00027-8, 2000.

[Widenius, 2002] Widenius, M. and Axmark, D., *The MySQL Reference Manual*, O'Reilly & Associates, Inc., ISBN: 0-596-002653, 2002. Also available on-line at: `http://www.mysql.com/documentation/index.html`.

[Zimmermann, 1996] Zimmermann, P., *The Official PGP User's Guide*, MIT Press, 1996.