ϖ A language based on the π-calculus

David Tunney, BSc
Supervisor Dr David Gray

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE
in the School
of
Computing
D C U

August 2005

Abstract

The exponential increase in the volume and sensitivity of data transmitted over electronic media has resulted in a corresponding increase in attempts to secure these inherently insecure transmissions Numerous networking protocols and associated mechanisms have been used but implementing distributed systems is a notoriously error prone exercise Attempts to ensure the relevant properties are present in distributed systems can be made by the application of formal methods However this application of formal methods is made to the specification of a distributed system, not its actual implementation Typically, a wide gulf exists between the specification of a distributed system and its actual implementation, and this gulf can result in the introduction of potentially devastating errors A method of bridging this gulf is required in order that the application of formal methods to distributed systems can become more widespread and more accessible We propose a general purpose programming language that is based on one of the more popular formal notations used to specify distributed systems, the π-calculus With this approach we allow the integration of complex sequential computations into π-calculus specifications of distributed systems to produce systems that are capable of execution in a distributed and concurrent fashion The implementation of this proposal is facilitated by designing the language such that fragments of Java code can be integrated into a π-calculus framework

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Masters of Science is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: [signature]

ID No.: 97479080

Date: 02/09/2005

Acknowledgements

This thesis and my Masters degree would not have been possible without the help, support and understanding of my family and of Glenda, nor without the never ending patience and guidance of my supervisor Dr Gray Many thanks to you all, I'm eternally grateful

Contents

List of Figures

Chapter 1

Introduction

Formal Methods are the applied mathematics of software development "Formal Methods bring to software and hardware design the same advantages that other engineering endeavours have exploited mathematical analysis based on models" (Butler, Caldwell, Carreno, Holloway & Miner 1995) The use of Formal Methods allows software developers to create models of the systems that they intend to implement and then to reason about this models - proving that certain properties hold true for the systems

Formal Methods, as the name suggests, are a broad and diverse collection of techniques that "are solidly based on mathematical logic systems and precise rules of inference" (Black, Hall, Jones & Windley 1996) A Formal Method usually consists of a language that can be used to describe the system that is to be implemented, as well as a set of axioms and rules that allow properties of the system to be proved While the concepts behind the various Formal Methods, and the notations that they use to describe the systems, vary massively from Formal Method to Formal Method, as does the domain for which the specific Formal Method is suitable, their purpose remains the same, "the fundamental goal of Formal Methods is to capture requirements, designs, and implementations in a mathematically based model that can be analysed in a rigorous manner" (Butler, Caldwell, Carreno, Holloway & Miner 1998)

The overall goal of the conception and application of Formal Methods, both as a group and individually, is to improve the standard of software delivered by increasing the ability of those implementing software systems to design their systems in sufficient detail and to prove that these systems possess the desired properties

While individual Formal Methods may be only very recently conceived, the concept of a Formal Method on the whole is not new Formal Methods have been around for some time and have yielded impressive results in fields such

as micro-chip design, aviation and aerospace (Butler et al 1995, Butler et al 1998, JPL n d , Heitmeyer 1998) amongst others Projects in which Formal Methods have been employed have seen reduced costs, increased reliability and more predictable delivery dates Taking into account the benefits that Formal Methods can bring to a project one could be forgiven for expecting that the application of Formal Methods would be the industry standard in the software development world This, however, is not the case - their application is the exception not the rule

The traditional reason for the non-application of Formal methods is cost (Hall 1990) It is widely held that it takes time, money and expertise to train people in the application of Formal Methods to software development and in the actual application itself With most software projects being highly cost and time sensitive Formal Methods are often overlooked in an attempt to keep within the constraints of a project's budget If, however, the application of Formal Methods in software projects was not as costly then, perhaps, they may not be as frequently overlooked

One such possible mechanism for reducing the difficulty, and therefore cost, of applying Formal Methods could, possibly, be to bridge the gap between the paradigms used in the design of the software systems and those that are used in the implementation of those systems Currently such a gap exists as the programming languages used in the implementation of software systems are largely procedural or object-oriented in nature while the Formal Methods used to specify the systems often are based on unique paradigms which are irreconcilable with those of the programming languages used in the implementation of the specified systems

1 1 Objectives

Proposed is a programming language that is based on a popular Formal Method - the π-calculus It is hoped that this programming language will exhibit all the recognisable qualities of the Formal Method in question in addition to being usable, robust, expressive and providing a high level of support for distributed computing It is desired that by possessing these properties that this programming language will make Formal Methods more accessible and increase their likelihood of being used

1.2 Outline of the thesis

- **Chapter one** Introduction to the research

- **Chapter two** Introduction to the π-calculus as well as variants and extensions of it
- **Chapter three** This chapter presents a number of existing related implementations
- **Chapter four** The design and theory of the programming language is presented in this chapter
- **Chapter five** The implementation behind the design and theory is explained in this chapter
- **Chapter six** This chapter contains a number of example systems implmented in this programming language
- **Chapter seven** The conclusions of this research

Chapter 2

Background: The π and Spi calculus

2.1 The π-calculus

In the late 1980s a unique form of distributed systems were becoming increasingly common-place and important, but the nature of these distributed systems differed significantly from the traditional distributed systems This new breed of distributed system was not static with regard to topology, it was continually changing Links between agents in the systems would grow and die in a seemingly organic fashion and these caused traditional modelling notations for distributed systems to struggle with this new strain of concurrent system This limitation of existing tools for modelling concurrent systems led Robin Milner, Joachim Parrow and David Walker to devise a process algebra called the π-calculus The π-calculus is heavily influenced by Milners earlier work on CCS (Milner 1989) and it retains many positive aspects of CCS and also adds, amongst other things, the notion of mobility Mobility being the ability of systems to grow and alter dynamically during their execution Due to the unique nature of the π-calculus, it is quite capable of capturing the essence of these systems that were dubbed *Mobile Systems* A mobile system tends to be distributed across a network and involves the concurrent execution of a number of agents, agents between who links can move

2 1 1 Introducing the π-calculus

As seemingly demanded by any modelling tool for mobile systems the basic computational operation in the π-calculus is the exchange of a communications link between processes It is this capability to send links from one process to

another that sets the π-calculus apart from other process algebras, such as CCS and CSP (Hoare 1985), which do not allow the communication of links, and it is only by receiving a link that a process can acquire a capability to interact with processes which were previously unknown to it In the example system, Figure 2 1, process A and process C are both connected to process B but not to each other, however they wish to be linked together This can be achieved by the creation of a link z by one of the processes, lets say process A After creation it is sent over x to process B who forwards it onto process C on y Now both process A and process C know of the link z and they can now interact with each other without an intermediary party An act of "learning" has taken place and two previously unconnected processes are now linked It is this moving of links that earns the π-calculus the title of *calculus of mobile systems*

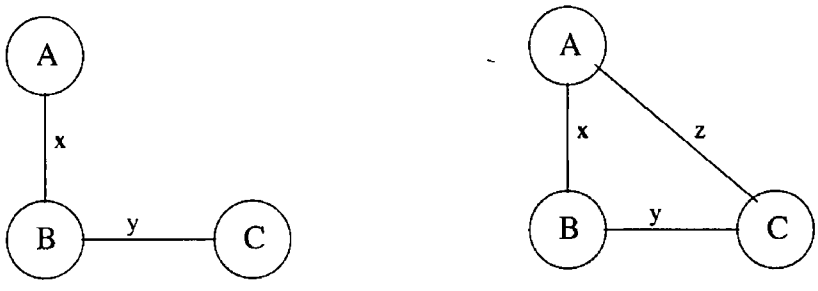

Figure 2 1 Learning processes

In the π-calculus there is the notion of a *name* and the intricacies of this notion contribute significantly to the expressive powers of the π-calculus A name is the most primitive entity in the π-calculus and is atomic in nature, i e it has no structure A name represents a link, or a channel, between processes but a name also is the data that is transmitted on these channels This dual nature of names is what allows the *extrusion of scope*[1] of names in the π-calculus, and it is this scope extrusion that allows new links to be learnt by processes

A π-calculus process can be thought of as a collection of π-calculus actions that combined achieve a specific task By grouping a number of processes together and allowing them to interact the π-calculus allows systems to accomplish their goals While this may seem very simplistic the π-calculus is more expressive than it first appears and it has been shown that the λ-calculus can be represented in its entirety within the π-calculus (Milner 1993)

When mobility and mobile computing are usually discussed the idea of mobile devices is what is generally thought of However the concept of mobility is not limited to devices that can be moved, mobility also covers the notion

[1] Scope extrusion is a concept vital to the π-calculus and will be explained later in this section

of a series of stationary devices, between which links grow, die and are passed about The π-calculus is capability of handling both forms of mobility, however its primary applications have been in scenarios similar to the latter

2 1 2 Syntax and Semantics

Syntax

It is assumed in the π-calculus that there is an infinite number of names and in the following section lower-case letters are used to represent names Also, while the π-calculus has no concept of process names, upper-case letters are used to indicate processes

There are a limited number of actions that a π-calculus process can perform, and these are collectively referred to as *action prefixes*, (α) Action prefixes can be assembled to form processes, and processes can be further assembled to form larger processes This construction of processes from action prefixes and larger processes from sub-processes is governed by the syntax of the π-calculus

Action Prefixes

α =	$x(z)$	Input prefix, z is received on x
	$\overline{x}z$	Output prefix, z is sent on x
	τ	An unobservable action

Processes

P =	0	Null process
	α P	Prefix
	P + Q	Sum
	P | Q	Composition
	[x==y]P	Match
	(vx)P	Restriction
	!P	Replication

- **Null process** The empty process, it cannot perform any actions
- **Prefix** The process P is prefixed by one of the valid prefixes - input, output or an unobservable action
- **Sum** Interaction can occur with *either* P or Q but not both Sum is often also referred to as the choice operator
- **Parallel composition** Represents the combined behaviour of the processes P and Q executing concurrently Processes running in parallel can interact with each other, or with third party processes, or a combination of both

- **Match** If x is equal to y then the process behaves as P, otherwise it blocks, i e does nothing
- **Restriction** $(vx)P$ behaves as P, but with the name x being local to P and only P
- **Replication** The process $!P$ is equivalent to $P \mid !P$ In other words $!P$ behaves as an infinite number of instances of P all executing in parallel to one another

Free and bound names

In both $x(y)\ P$ and $(\nu y)P$ both the names x and y are bound within the scope of P A name is said to be bound in a process if it is either (i) a binding occurrence, i e $x(y)$ or (νy), or (ii) it is within the scope of a binding occurrence A bound name can only be referenced from within the scope of its binding occurrence and as such it cannot be used to communicate with a process that lies outside this binding occurrence, unless its scope is extruded to include that process

A name is a free name if it is not a bound one The free names of a process, P, are denoted by fn(P), and the bound names are denoted bn(P)

When interaction occurs between processes a ***substitution*** generally occurs A substitution is a function from names to names $\{z/y\}$ indicates a substitution that replaces y with z and leaves all other names untouched

e g $x(y)\ P \mid \overline{x}z\ 0 \xrightarrow{\overline{x}z} P\{z/y\}$

$P\{z/y\}$ behaves as P with all occurrences of y replaced by z, alpha-conversion of already existing occurrences of z may be necessary Alpha-conversion being the conversion of a process by renaming elements of that process in a consistent manner

Structural Congruence

It is necessary to be able to equate processes that differ only in terms of organisation A method of identifying processes which represent the same computations is required, i e $\overline{a}y\ 0 \mid a(x)\ 0$ and $\overline{b}y\ 0 \mid b(x)\ 0$ are intuitively the same and should be identified as such This is achieved in the π-calculus via *structural congruence*, $\equiv$ Structural congruence identifies only processes where it is clear from their structure that they are the same

Structural congruence is defined as the smallest congruence that satisfies the following rules

1 If Q can be obtained by alpha-conversion of P, then $P \equiv Q$

2 (a) Commutativity of parallel composition, $P \mid Q \equiv Q \mid P$

(b) Associativity of parallel composition, $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$

(c) Commutativity of sum, $P + Q \equiv Q + P$

(d) Associativity of sum, $(P + Q) + R \equiv P + (Q + R)$

3 *Scope extrusion laws*

$(\nu x)0 \equiv 0$

Intuitively these two terms are equivalent as since there is nothing to restrict in *0* then the presence of a *restriction operator cannot have* an effect on *0*

$(\nu x)(P \mid Q) \equiv P \mid (\nu x)Q$ if $x \notin fn(P)$

If a name is not free in a process then the act of restricting, or not restricting, the name on that process will not have an effect on that process

$(\nu x)(P + Q) \equiv P + (\nu x)Q$ if $x \notin fn(P)$

If a name is not free in a process then the act of restricting, or not restricting, the name on that process will not have an effect on that process

$(\nu x)[a{=}{=}y]P \equiv [a{=}{=}y](\nu x)P$ if $x \neq a$, $x \neq y$

If a restriction does not operator on elements of a match operator then the positioning of the restriction relative to the match will not have an impact of the behaviour of the match operation

Semantics

In the same manner as most process algebra the *operational semantics* of the π-calculus is given via a reduction semantics The following semantics are specified using reduction semantics Reduction semantics is a method of formal semantic specification which works by transforming complex expressions into simpler ones Each step in this process is called a 'reduction' and once an expression is fully reduced and the reduction process has terminated then the expression is said to be in its normal form A complex expression is equivalent to its reduced form, and this reduced form is simpler to reason about

The rules of this reduction semantics are

[Struct] $$\frac{P' \equiv P,\ P \xrightarrow{\alpha} Q,\ Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$$

[Prefix] $$\frac{}{\alpha P \xrightarrow{\alpha} P}$$

[Par] $$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

[Sum] $$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

[Com] $$\frac{P \xrightarrow{x(y)} P',\ Q \xrightarrow{\overline{x}z} Q'}{P \mid Q \xrightarrow{\tau} P'\{z/y\} \mid Q'}$$

[Res] $$\frac{P \xrightarrow{\alpha} P',\ x \notin \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$$

[Match] $$\frac{P \xrightarrow{\alpha} P'}{[x==x]P \xrightarrow{\alpha} P'}$$

[Rep] $$\frac{}{!P \longrightarrow P \mid !P}$$

Explanations

1 [**Struct**] If the occurrence of an action causes the process P to reduce to the process Q, then a process that is structurally congruent to P can be reduced to a process that is structurally congruent to Q on the occurrence of the same action

2 [**Prefix**] A process that is prefixed by an action reduces to that process after the occurrence of the specific action

3 [**Par**] If a process, P, can reduce to another process, P', after the occurrence of an action then P can reduce to P' regardless of what processes are running concurrent to it when that action, α, occurs

4 [**Sum**] If a process, P, can reduce to another process, P' after the occurrence of an action then the sum of P and any other processes will reduce to P' on the occurrence of α

5 [**Com**] If a process P reduces to P' on an input action on a specific name and if the process Q reduces to Q' on an output action on that same

name then P in parallel to Q will reduce to P' in parallel to Q' after an unobservable action occurs

6 [**Res**] If P reduces to the process P' on an action, and the name x is not involved in this action, then the reduction will still occur if the name is restricted in both processes

7 [**Match**] If a process reduces to another process on an action, then this reduction will still occur if the names being compared are the same Otherwise nothing will happen

8 [**Rep**] A replicated process reduces to that same replicated process with a non-replicated instance of the same process in parallel

2 1 3 Basic Examples

Consider the process

Simple I/O $\quad \overline{x}a\ 0 \mid a(b)\ \overline{b}v0 \mid x(y)\ \overline{y}z\ 0$

The above process is comprised of three sub-processes, and although the π-calculus processes are not named for this explanation we will refer to them as P, Q and R, as read from left to right In all three sub-processes the names x and a are free, all occurrences of x and a in the three sub-processes all refer to the same names It is these free names that allow the interaction of P and R (over the name x) to occur, likewise for the interaction of P and Q (over the name a

The act of reading a name over another name is said to *bind* that name in the process that follows the input action For example in the process R the action $x(y)$ binds the name y in the remainder of the process, $\overline{y}z\ 0$ In reality the name y will never actually be used in the remainder of the process as y is only a placeholder that indicates where the name read in on the channel x in the action $x(y)$ should be substituted in the process

It is worth noting at this stage that in the original π-calculus the act of exchanging a name over a channel is a synchronous one - for every input there must be a corresponding output and vice versa and without the corresponding action any attempt to input or output will simply block until there is a corresponding action

In order for the process to reach its final state there must be a certain amount of interaction between processes These interactions proceed as follows

$\overline{x}a\ 0 \mid a(b)\ \overline{b}\ v\ 0 \mid x(y)\ \overline{y}z\ 0 \quad \xrightarrow{\tau}$
$0 \mid a(b)\ \overline{b}v\ 0 \mid \overline{a}z\ 0 \quad \xrightarrow{\tau}$
$0 \mid \overline{z}v\ 0 \mid 0$

The reason that the above processes could interact successfully is that they shared a certain amount of knowledge of names If the information that was shared was restricted between specific processes then the execution would have been quite different The following process is virtually identical to the previous one except that the name x is restricted to the process P and Q, that is the name x that appears in P and Q is a different x to the one in process R

Restriction $(\nu x)(\overline{x}a\ 0 \mid a(b)\ 0\ ) \mid x(y)\ \overline{y}z\ 0$

Since the occurrence of x in $P|Q$ is a different x to the one in R no interaction is possible This restriction of names allows the creation of private channels between processes, a feature that proves invaluable for when dealing with concurrent, distributed systems The real value of how the π-calculus handles restriction of names is only appreciated when one considers the unique concept of scope extrusion

Scope extrusion $(\nu y)\ ((\nu x)\ (x(m) \mid \overline{y}x) \mid y(p)\ \overline{p}o)$

In the above example the process $x(m)$ can be considered as a resource, the process $\overline{y}x$ can be considered an access control unit for the resource and the process $y(p)\ \overline{p}o$ can be viewed as an agent wishing to access the resource, Figure 2 2

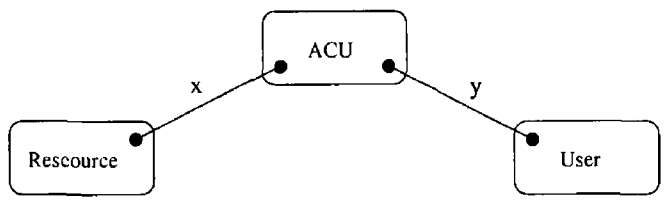

Figure 2 2 Resource Access Control

The resource in question can only be accessed via the name x, the only entities that initially are included in the scope of this name are the ACU and the resource itself but by sending the name x over the name y the scope of the name x can be extruded to include the User process The User process can now access the resource in question, Figure 2 3

Scope extruded $(\nu y)(\nu x)(x(m) \mid 0) \mid \overline{x}o)$

The notions of access control and resources play a major part in concurrent

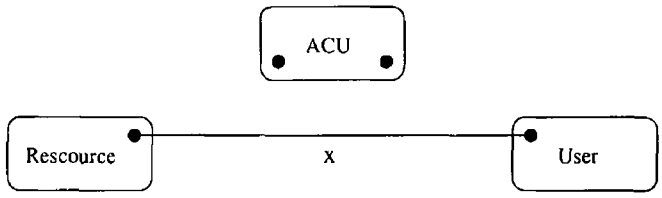

Figure 2 3 Resource Access Control

computing and the ease with which the π-calculus allows these ideas to be expressed highlights once again the benefits of using the π-calculus when dealing with concurrent and distributed systems

In any real world concurrent system there is always a possibility that either one thing or another will happen, that a choice will be needed to be made between certain actions Choices like these can be expressed in the π-calculus via the choice (+) operator

Choice $P + Q$

In the above process either process P or Q is started and the choice of which process to start is made in a non-deterministic fashion, that is the result of the operation cannot be predicted before its execution Choosing a process in a completely arbitrary manner is of limited use and as a result it is much more common to see a guarded choice expression

Guarded Choice $\overline{x}a\ P + \overline{y}a\ Q + w(b)\ R$

Once again the above expression will result in either process P, Q or R being started, however in this case the choice is not a non-deterministic one, but rather is based on which action occurs first $\overline{x}a$, $\overline{y}a$ or $w(b)$ Whichever action completes first results in the associated process being started A guarded choice expression consists of an arbitrary number of possible branches of execution, or choice options For example in $\overline{x}a\ P + \overline{y}a\ Q + w(b)\ R$ there are three possibilities $\overline{x}a\ P$, $\overline{y}a\ Q$ and $w(b)\ R$ Exactly one of these options must be executed, and interaction between options is not permitted, i e in the process $x(a)\ P + \overline{x}b\ Q$, the reduction to $\{b/a\}P + Q$ cannot occur

Should the same action occur in more than one place in an expression, a non-deterministic choice is made between these options should the associated action prove successful Any combination of input and output actions are valid as prefixes to processes in a choice expression and these prefixes are said to "guard" the respective processes

A concurrent system can sometimes involve multiple copies of the same pro-

cess running in parallel Sometimes the number of instances may be an arbitrary one, one that cannot be known in advance, and a mechanism to capture this behaviour is required This mechanism comes in the form of the replication operator, (!) A process that is replicated behaves as if there are an infinite number of copies of the process ready to run at any stage, once one starts another is immediately ready to start A replicated process $!P$ can be expanded to $P|!P$

Replication(1) $\quad !\overline{x}a \mid x(b)\ P \mid x(c)\ Q$

The execution of the above process could proceed as

$!\overline{x}a \mid x(b)\ P \mid x(c)\ Q \longrightarrow$
$\overline{x}a \mid !\overline{x}a \mid x(b)\ P \mid x(c)\ Q \longrightarrow$
$\overline{x}a \mid \overline{x}a \mid !\overline{x}a \mid x(b)\ P \mid x(c)\ Q \xrightarrow{\tau}$
$0 \mid \overline{x}a \mid !\overline{x}a \mid P\{a/b\} \mid x(c)\ Q \xrightarrow{\tau}$
$0 \mid 0 \mid !\overline{x}a \mid P\{a/b\} \mid Q\{a/c\}$

Think back to the access control example, it only worked for one connection from a user to the resource, this is not a realistic system A more realistic system is one in which multiple users want to make multiple separate connections to the resource

Replication(2) $\quad (\nu y)(!(\nu x)(x(m)\ P \mid \overline{y}x) \mid (\nu o)(!y(p)\ \overline{p}o))$

Multiple users, $!y(p)\ \overline{p}o$, are now able to access the resource by means of multiple, access control process/resource pairings, $!(\nu x)(x(m) \mid \overline{y}x)$ Each time a user process kicks off another one is ready to take its place Each replicated user process receives its own private channel for communicating with the resource as the entire access control/resource process is replicated including the restriction As a result the system can now handle multiple users making multiple connections to the resource

$(\nu y)(!(\nu x)(x(m)\ P \mid \overline{y}x) \mid !(\nu o)(y(p)\ \overline{p}o))$
$\longrightarrow$
$(\nu y)((\nu z)(z(m)\ P \mid \overline{y}z) \mid !(\nu x)(x(m)\ P \mid \overline{y}x) \mid !(\nu o)(y(p)\ \overline{p}o))$
$\longrightarrow$
$(\nu y)((\nu z)(z(m)\ P \mid \overline{y}z) \mid !(\nu x)(x(m)\ P \mid \overline{y}x) \mid (\nu q)(y(p)\ \overline{p}q) \mid !(\nu o(y(p)\ \overline{p}o))$
$\longrightarrow$
$(\nu y)((\nu z)(z(m)\ P \mid 0) \mid !(\nu x)(x(m)\ P \mid \overline{y}x) \mid (\nu q)(\overline{z}q) \mid !(\nu o(y(p)\ \overline{p}o))$
$\xrightarrow{\tau}$
$(\nu y)(\nu z)((\nu q)(P\{q/m\} \mid 0) \mid !(\nu x)(x(m)\ P \mid \overline{y}x) \mid 0 \mid !(\nu o(y(p)\ \overline{p}o))$
$\xrightarrow{\tau}$
$(\nu y)(\nu z)((\nu q)(P\{q/m\}) \mid !(\nu x)(x(m)\ P \mid \overline{y}x) \mid !(\nu o(y(p)\ \overline{p}o))$

2 1 4 Example

Imagine a system involving an arbitrary number of users which can access an arbitrary number of printers Users wish to print only one job at a time and do not care which printer performs the task Printers can only handle one job at a time and following completion of one job are ready to print another one Users must pay per print job and billing is performed by routing all print requests through a central access control unit (ACU)

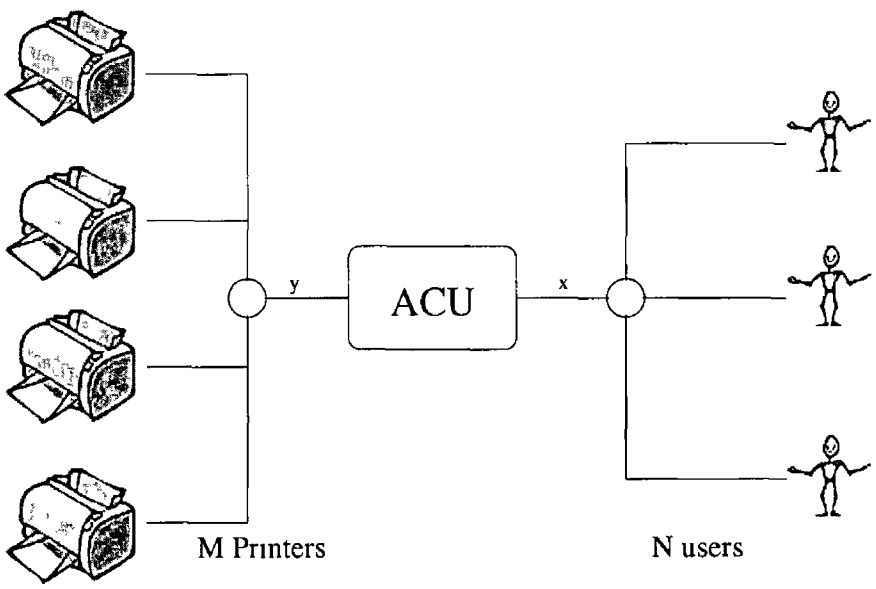

Figure 2 4 Printer example

The following is a π-calculus specification for the above system

$$System = (\nu y)(\nu x)(!User \mid ACU \mid \overset{n-1}{\Pi}_{i=0} Printer(i))$$
$$User = (\nu a)(\nu k)(\overline{x}a\ a(c)\ \overline{c}k\ 0)$$
$$Printer(j) = (\nu q)(\overline{y}q\ q(e))\ Printer(j)$$
$$ACU = x(b)\ y(c)\ \overline{b}c\ ACU$$

This example introduces a syntactic shortcut of the π-calculus, $\overset{n-1}{\Pi}_{i=0}$ In this example there will be n printers running in parallel in the system, and rather than writing $(Printer(1) \mid Printer(2) \mid \ \mid Printer(n))$ it is much more convenient to write $\overset{n-1}{\Pi}_{i=0}$, which is merely shorthand for the more lengthy and complex expression involving n printers in parallel being explicitly described Another form of syntactic sugaring is also introduced in this example, that is the use of parameterised processes, e g *Printer(j)* Parameterisation of processes

allows a generic definition of a process to be used in many specific cases where the only differences between the instances of the process is the value of the parameter(s) supplied to it

Another new concept introduced in this example is the notion of tail recursion, e g $Printer(y) = (\nu q)(\overline{y}q\ q(e))\ Printer(y)$ The occurrence of *Printer* at the end of this expression is referred to as tail recursion Tail recursion captures the idea that once a process has finished executing it may be required to return to its original state and be ready to execute again

The system consists of three entities - users, printers and an Access Control Unit (ACU) Users cannot immediately interact directly with printers, they must first go through the ACU A channel x is shared by the User processes and the ACU, likewise another channel y is shared by Printer processes and the ACU User processes in the System process are replicated to reflect the possibility of many users

The first thing that a User process does is extend the scope of the name a to include the ACU by sending this name over the channel x This name will be used for all future communications with the ACU Next the user reads another name in over the channel a This new name, c, will be used to transmit the print job, represented by the name k, to the printer, as the name c is known to both the printer and the user

A Printer process creates a name q, this name is the name that will ultimately be used to receive the print job from the user Once a print job is received it is deemed to be printed A printer process sends the name q to the ACU via the shared name y It then waits to receive a name on the channel q This name, once received, is the print job that should be printed Once this print job is printed the printer is ready to receive more jobs

The ACU process is designed to allow users and printers to eventually interact It reads the relevant information from both users and printers and passes the relevant information onto the correct parties

Printer example reduction

Let $n = 2$

$$(\nu x)(\nu y)(!User \mid ACU \mid printer(1) \mid printer(2))$$
$$\longrightarrow$$
$$(\nu x)(\nu y)((\nu wp)(\overline{x}w\ w(c)\ \overline{c}p\ 0) \mid !User \mid x(b)\ y(c)\ \overline{b}c\ ACU$$
$$\quad \mid (\nu m)\overline{y}m\ m(e)\ printer(1) \mid printer(2))$$
$$\xrightarrow{\tau}$$
$$(\nu x)(\nu y)((\nu wp)(w(c)\ \overline{c}p\ 0) \mid !User \mid y(c)\ \overline{w}c\ ACU$$
$$\quad \mid (\nu m)\overline{y}m\ m(e)\ printer(1) \mid printer(2))$$
$$\xrightarrow{\tau}$$
$$(\nu x)(\nu y)((\nu wp)(w(c)\ \overline{c}p\ 0) \mid !User \mid (\nu m)(\overline{w}m\ ACU$$
$$\quad \mid m(e)\ printer(1) \mid printer(2)))$$
$$\xrightarrow{\tau}$$
$$(\nu x)(\nu y)((\nu wpm)(\overline{m}p\ 0) \mid !User \mid ACU$$
$$\quad \mid m(e)\ printer(1) \mid printer(2))$$
$$\xrightarrow{\overline{w}m}$$
$$(\nu x)(\nu y)(!User \mid ACU \mid printer(1) \mid printer(2))$$

2.2 Extensions and variations of the π-calculus

Following the publication of the initial research on the π-calculus many academics recognised the benefits of the π-calculus and much further research was undertaken into the π-calculus, a fact which is quickly apparent from the sheer volume of publications now available on the field A large proportion of this research involved the creation of variants and extensions of the π-calculus The creators of these variants and extensions felt that the modifications that they made, which were sometimes major and sometimes minor, either did the same as the π-calculus only better/neater/quicker/etc or that the π-calculus was a sub-set of their creation which did all that the π-calculus could and more Many such extensions exist, some were short-lived and not widespreadly recognised or adopted by some become more established Four of the more well known extensions/variants are the Fusion calculus (Parrow & Victor 1998), the Join calculus (Fournet & Gonthier 1996), the Ambient calculus (Parrow & Victor 1998) , and the Spi calculus

2 2 1 The Fusion calculus

The Fusion calculus is an extension to the π-calculus that was devised by Parrow and Victor (Parrow & Victor 1998) The goal of the Fusion calculus was to create an extension to the π-calculus that simplifies the π-calculus and allows simpler modelling of systems that involve shared state For the main part this was achieved by the addition of a mechanism which allows the updating and maintaining of state and enforces symmetry between input and output actions This mechanism is provided by the addition of a new class of action called "fusions" The Fusion calculus also proved popular and it too has spawned extensions and variations of its own

The Join Calculus

Fournet and Gonthier (Fournet & Gonthier 1996) aimed to create an extension of the π-calculus that retains the expressivity of the π-calculus, more specifically the asynchronous summation free π-calculus This extension was to be more amenable than the π-calculus to being used as the basic of a complete implementation of a distributed programming language They believed that the π-calculus in its original form was not implementable, and that it would be necessary to bring the specification notation down a few levels closer to that of a programming language before it could be implemented The Join calculus also proved to be popular and extensions and implementations of it exist

2 2 2 The Spi calculus

The Spi calculus was created by Abadi and Gordon (Abadi & Gordon 1997) as an extension to the π-calculus that was specifically designed with the task of implementing security protocols in mind They believed that the inclusion of cryptographic primitives and operations in the syntax and semantics of the π-calculus would make the Spi calculus much more capable of modelling security related systems

2 2 3 The Ambient calculus

The creators of the Ambient calculus, Cardelli and Gordon (Cardelli & Gordon 1998), believed that existing process calculi were neglecting what they felt was a central concept of mobility - the mobility of processes They believed that the ability for agents to migrate from location to location was key to any mobility orientated process calculi, and that this migration should occur in a clearly defined and controlled manner This migration is provided by a construct which represents location - the ambient

While a detailed examination of all, or even some, of the π-calculus extensions and variations that exist is beyond the scope of this document, a detailed examination of one such extension will be given for a sense of completeness This extension is the Spi calculus There were a number of reasons why the Spi calculus was chosen to be examined in more depth

1 The Spi calculus features the addition of sequential computations, albeit limited sequential computations, to the syntax and semantics of the π-calculus

2 With the exception of the addition of the mechanism for performing sequential computations the syntax and semantics of the Spi calculus is largely identical to that of the π-calculus

3 The Spi calculus is used to model systems involving security protocols, the implementation of security protocols is ultimately one of the target uses of ϖ

2 3 The Spi calculus

2 3 1 Introducing the Spi calculus

The Spi calculus is an extension of the π-calculus The primary purpose of the Spi calculus is the description and analysis of cryptographic protocols While the π-calculus allowed an abstract overview of a protocol, the Spi calculus allows

a much more detailed view of a cryptographic protocol To facilitate this more detailed approach to cryptography a full complement of cryptographic primitives are provided in the Spi calculus, including

Symmetric encryption Symmetric encryption is the encryption and decryption of data using a secret key The same key is used both for the encryption and the decryption

Asymmetric encryption Asymmetric encryption involves the use of key pairs for the encryption and decryption of data Data encrypted with one part of a key pair can only be decrypted with the corresponding part of that pair, and vice versa One part of the key pair is kept secret, the "private" key, while the other is freely distributed, the "public" key

Hashing A hash function is a mathematical one-way function When data is hashed a cryptographically unique value of a fixed length is acquired This value is different for each different input data, and the same input data will always yield the same hash

For the purposes of this document the Spi calculus is, in essence, identical to the π-calculus bar the addition of cryptographic primitives to the syntax of the π-calculus The concepts of names, channels and processes remain the same and any valid π-calculus specification is also a valid Spi calculus one

The Spi calculus is particular suited to specifying security protocols as its "model of protocols takes into account the possibility of attacks but does not require writing explicit specifications for an attack"(Abadi & Gordon 1997) Anyone familiar with security protocols and formal methods will instantly recognise the benefits of this property of the Spi calculus, by avoiding the need to explicitly define the capabilities of attackers one avoids the dangers of missing capabilities of the attacker This property combined with the rest of the properties of the π-calculus make the Spi calculus a very useful tool for describing security protocols

2 3 2 Syntax additions

The syntax of the Spi calculus is virtually identical to that of the π-calculus except for the addition of cryptographic primitives These additions come in two forms, those concerned with terms, and those concerned with processes

In the π-calculus there are only names, operations can only be performed on names, however this is not the case in the Spi calculus While the π-calculus refers to names, the Spi calculus refers to terms, where a term is

M,N,L	=	n	a name
		$\{M\}_n$	Symmetric encryption
		H(M)	Hashing
		M^+	The public part of a key-pair
		M^-	The private part of a key-pair
		$\{\|M\|\}_N$	Asymmetric encryption
		$[\{M\}]_N$	Private-key signature

The other addition to the syntax is the set of actions that can prefix a process

α =

case L of $\{x\}_N$ in P	Symmetric decryption
case L of $\{\|x\|\}_N$ in P	Asymmetric decryption
case N of $[\{x\}]_M$ in P	Signature checking

1 **Symmetric encryption** If a process wishes to send some data, M, to another process and it requires that the data is encrypted using some sort of symmetric cipher under a specific key, n, this is represented by writing $\{M\}_n$

Example $d(m)\,\overline{c}\{m\}_k$

The name m is read over the channel d This name is encrypted using a symmetric cipher with the name k as the key, and the cipher-text resulting from this operation is then sent over the channel c

2 **Hashing** The result of hashing, or digesting, a name is term H(M)

Example $d(m)\,\overline{c}(H(m))$

The name m is read over the channel d This name is then hashed and the resulting hash is send over the channel c

3 **Public key** A key-pair comprises of a shared and a secret part, or a public and private part A key-pair in the Spi calculus is represented by a single name so some method of accessing both parts is needed

Example $(\nu k)(\overline{d}k^+)$

A key-pair k is created and then the public part of that key pair, k^+, is sent over the channel d

4 **Private key** Both parts of a key pair need to be accessed even though the public part is generally the only part that is passed around

Example $((\nu k)(\overline{d}k^{-})$

A key-pair k is created and then the private part of that key pair, k^{-}, is sent over the channel d Care should be taken that the private parts of keys are kept secret

5 **Asymmetric encryption** $\{|M|\}_N$ represents the encryption of the term M under the public key N The result of this action can be sent and received on channels as any other name could be

Example $d(m)\,\overline{c}\{|m|\}_k$

The name m is read over the channel d and is then encrypted with the public-key k and the resulting cipher-text is transmitted over the name c

6 **Private key signature** The signing of data involves the encryption of the data using the private-key This data can only be decrypted by using the associated public-key and the act of successful decrypting the data using the public-key ensures that it was, in fact, "signed" by the relevant party

Example $d(m)\,\overline{c}[\{m\}]_k$

The name m is read in over the channel d, this name is then signed using the private key k and the resulting signature is then sent over the channel c

7 **Asymmetric decryption** In *case* M *of* $\{|x|\}_N$ *in* P, if M is in fact the result of encrypting a name with the relevant public-key then it will be of the form $\{|O|\}_N$, this value of O, i e the decrypted data, is substituted for all occurrences of x in P If it is not then this action blocks indefinitely

Example $d(m)$ *case* m *of* $\{|x|\}_n$ *in* $\overline{d}x$

The term m is read in over the name d, an attempt is made to decrypt the name using the key n, if this attempt succeeds then the resulting plain-text is sent over the name d

8 **Symmetric decryption** In *case* M *of* $\{x\}_k$ *in* P, if M is in fact the result of encrypting a name with the key k then it will be of the form $\{O\}_N$, this value of O, i e the decrypted data, is substituted for all occurrences of x in P If it is not then this action blocks indefinitely

Example $d(m)$ *case* m *of* $\{x\}_n$ *in* $\overline{d}x$

The name m is read in over the name d, an attempt is made to decrypt the name using the key n, if this attempt succeeds then the resulting plain-text is sent over the name d

9 **Signature checking** Checking a signature is much like asymmetric decryption except that the key used to decrypt the data is a public key rather than a private key Bar this difference, and the syntactic difference, signature checking is the same as asymmetric decryption

2 3 3 Cryptographic assumptions

The creators of the Spi calculus made some significant, yet reasonable assumptions with regard to cryptographic primitives and operations

- For data encrypted with a symmetric cipher, it is assumed that the only way to decrypt that data is to know the correct key
- For data encrypted with an asymmetric cipher, it is assumed that the only way to decrypt that data is to know the corresponding private key
- That sufficient redundancy is present in messages so that it can be detected if a cipher-text was encrypted with a specific key
- That the data used to create a hash cannot be recovered from the hash
- That no two distinct pieces of data will yield the same hash
- That a private-key cannot be obtained from its public-key

2 3 4 Operational semantics

The semantics of the π-calculus are a sub-set of the semantics of the Spi calculus and everything valid in the π-calculus is also valid in the Spi calculus There are, however, three additional rules in the operational semantics of the Spi calculus

[SymDec] $$\frac{L = \{M\}_N}{case\ L\ of\ \{x\}_N\ in\ P \longrightarrow P\{M/x\}}$$

[AsymDec] $$\frac{L = \{|M|\}_N{}^+}{case\ L\ of\ \{|x|\}_N{}^-\ in\ P \longrightarrow P\{M/x\}}$$

[SigCheck] $$\frac{L = [\{M\}]_N{}^-}{case\ L\ of\ [\{x\}]_N{}^+\ in\ P \longrightarrow P\{M/x\}}$$

Explanations

1 [**SymDec**] If a term, L, is the result of encrypting the term, M, with a symmetric cipher and the key N, then the result is the process P, with all occurrences of x replaced by M

2 [**AsymDec**] If a term, L, is the result of encrypting the term, M, with an asymmetric cipher such as RSA and the public part of the key pair N, the result is the process P with all occurrences of x replaced with M

3 [**SigCheck**] If a term, L, is the result of the term M being signed with the private part of the key-pair N then the process P continues with all occurrences of x being replaced by M

2 3 5 Example

Consider the printer example given in section 2 1 4, the π-calculus example Suppose that it was a requirement, for whatever reason, of this system that all jobs sent from users to printers must be encrypted The following is a Spi calculus system that achieves this

$$System = (\nu K^{+}_{ACU})(\nu xy)(!User \mid ACU \mid \prod_{i=0}^{n-1} Printer(i))$$

$$User = (\nu aj K_u)(\overline{x}\{|a|\}_{K^{+}ACU}\ \overline{a}\{|K_u|\}_{K^{+}ACU}\ a(c)$$
$$case\ c\ of\ \{z\}_{K_u}\ in\ \overline{z}\{j\}_{K_u}\ 0$$

$$Printer(j) = (\nu q K_p)(\overline{y}\{|q|\}_{K^{+}ACU}\ \overline{q}\{|K_p|\}_{K^{+}ACU}\ q(d)$$
$$case\ d\ of\ \{K_p\}_{K_p}\ in\ q(e)\ \ case\ e\ of\{p\}_{K_u}\ in\ 0$$

$$ACU = (\nu K^{-}_{ACU})(x(m)\ m(n)casenof\{|K_m|\}_{K_{ACU}}in$$
$$y(d)\ d(e)caseeof\{|K_d|\}_{K_{ACU}}in$$
$$\overline{d}\{K_m\}_{K_d}\ \overline{m}d\ ACU)$$

The topology of this system remains unchanged, the different processes are connected in the same way and they learn of channels in the same manner and order However, the interaction between the different processes is significantly more complex as keys and encrypted data are exchanged in an effort to ensure secrecy

At the top level the system has only one change - all User and Printers now know the public key belonging to the ACU This will allow all Users and Printers to encrypt their transmission to the ACU and allow the secure exchange of keys for use with symmetric ciphers

An instance of the User process wants to set up a channel and a key that will be used to transmit securely the print job to the printer This set-up is achieved by sending a channel to the ACU, and by then sending a session key, also encrypted, to the ACU on this channel It then receives a channel which has been encrypted with the session key and it is on this channel that the encrypted print job will be sent directly to the printer

A printer process also wants to establish the session key and a channel on which it will receive the print job It also does this by sending a channel and a session key, both encrypted, to the ACU It then receives back from the ACU another session key which is encrypted with the key that was sent to the ACU It is this session key that will be used to decrypt the print job once it is received on the channel that was sent to the ACU Once a printer has dealt with the received print job it returns to its initial state, ready to complete the procedure all over again

The ACU process is easily the most complex of all the processes in the system as it has to interact with both User and Printer processes to facilitate the secure exchange of channels and the establishment of session keys

The first step of the ACU process is to receive a name from the User process Using this name another name is read from the User This name is the session key for this user session that was generated by the User and was encrypted using the public key of the ACU Once this session key has been received and decrypted the ACU reads a name from the Printer process Similarly another name is read then from the Printer using this name This name is the session key for the printer session Using this printer session-key the user session-key is encrypted and then sent on the channel that is shared between the ACU and the printer Once this occurs this same shared name is sent to the User process so that the User process can communicate with the Printer

Once the ACU process is finished the ACU process also returns to its initial state ready to facilitate more transactions between users and printers

2.4 Conclusions

By now the expressive capabilities of the π-calculus should be clear A large range of powerful constructs and operations are available in the π-calculus However concepts such as replication, channels and the π-calculus approach to interaction between concurrently execution processes, which are simple and transparent to use in the π-calculus are not present in conventional programming languages and would prove rather cumbersome and troublesome to implement and use in these conventional programming languages While these absent elements could possibly be written as components and plugged into some of the conventional programming languages and the functionality of the constructs may be approximated, an unbridgeable gulf between the syntax and semantics of the π-calculus and those of the conventional programming language even with the added functionality would still exist This gulf and the problem that it poses for the task of comparing π-calculus specifications with their implementations in conventional programming languages creates a niche for programming

languages which are based directly on the π-calculus and its derivatives

2.5 Further Reading

The π-calculus

The theory behind the π-calculus is massive This introduction is intended only to give a brief overview of the syntax, semantics and purpose of the π-calculus and entire sections of the π-calculus have been omitted as they are beyond the scope of this document In order for any reader to get a true understanding of the π-calculus it would be necessary to read one or more of the following texts (increasing complexity)

- An Introduction to the pi-Calculus(Parrow 2001)
- The Polyadic pi-Calculus A Tutorial(Milner 1993)
- Communicating and Mobile Systems The Pi-calculus(Milner 1999)
- A Calculus of Mobile Processes Parts I and II(Milner, Parrow & Walker 1989)
- The π-calculus A Theory of Mobile Processes(Sangiorgi & Walker 2001)

The Spi calculus

As the Spi calculus is based upon the π-calculus a full understanding of the π-calculus is required before moving on to the Spi calculus The theory behind the Spi calculus is considerable and a lot of the important aspects of the Spi calculus haven't even been mentioned in this document If a reader desired a greater knowledge of the Spi calculus the following texts would be a good place to start

- A Calculus for Cryptographic Protocols The Spi Calculus(Abadi & Gordon 1997)
- A Calculus for Cryptographic Protocols The Spi Calculus(SRC Tech report)(Abadi & Gordon 1998)

Chapter 3

Related Research

Following the publication of the initial research on the π-calculus many derivatives of the π-calculus quickly emerged (Cardelli & Gordon 1998, Parrow & Victor 1998, Fournet & Gonthier 1996), and after some time implementations of the π-calculus, and these derivatives, began to appear While all of these implementations are, in some way, each unique with regard to how they *approach* the implementation of their underlying process calculus, it is possible to group the vast majority of these implementations into one of three categories based on various classification criteria Most of these implementations are based on the π-calculus, although some were inspired by more exotic variants or extensions of the π-calculus

The creation of these classification criteria occurred as the research into related work was taking place It was felt that these particular classification criteria would allow the fundamental differences between implementations to be determined and for the implementations to be subsequently grouped accordingly into categories Following the examination of related research it became clear that these classification criteria resulted in implementations falling into one of three categories Before stating what the three categories are the classification criteria will be examined and justified The classification criteria for these implementations are

Syntax

When inspecting the syntax of a language that is supposedly based on the π-calculus the primary concern is whether or not the syntax is similar to that of the π-calculus If they are similar there will be a visual likeness between the π-calculus and the implementation, i e they will look the same It was felt that this was important so that a comparison between specification and implementation would be as simple as possible and without a

need for translation

Semantics

An examination of the semantics of a π-calculus inspired language should, ideally, reveal a significant resemblance to those of the π-calculus The closer the semantics of a language to the semantics of the π-calculus, the more similar the behaviour of the language will be to the behaviour of the π-calculus, i e they will act the same The importance of this property is due to the desire to simplify the comparison process

Mobility

The π-calculus concept of mobility is just one approach to mobility and various implementations incorporate a different process calculus concept of mobility rather than that of the π-calculus How an implementation handles mobility has a significant impact on its ties to the π-calculus

Synchronous vs asynchronous communications

The communication of data over channels can be done in one of two ways, in a synchronous fashion, or in an asynchronous manner Versions of the π-calculus exist that are either synchronous or asynchronous in nature Similarly implementations of the π-calculus differ on this depending on which version of the π-calculus they are based on

Distribution

Some implementations of the π-calculus were designed to be used in the implementation of distributed systems and as such constructs, operators and environmental features were provided to allow this distribution Some implementations were only meant for use in systems whose execution would occur entirely on one host These distribution oriented languages are closer in spirit to the π-calculus The absence/presence of support for distribution is an easily determined classification criterion, and given that the π-calculus is intended for use in modelling distributed systems, an important one

Sequential computations

The π-calculus does not provide a mechanism for performing complex sequential computations such as cryptographic operations or even text manipulation Such a mechanism is necessary to be present in an implementation of the π-calculus for that implementation to be of use in a real world scenario, however such a mechanism is not always provided by implementations of the π-calculus

The majority of implementations fall into one of three main categories when classified using the above criteria

Category 1

A programming language belonging to category one is capable of performing complex sequential computations in a simple and transparent manner However is not designed for use in implementing distributed systems and it is not strictly based on the π-calculus

Category 2

A category two programming language is syntactically and semantically similar to the π-calculus but it is not capable of performing sequential computations in a simple fashion nor is it intended for use in implementing distributed systems

Category 3

Category three programming languages are syntactically and semantically similar to the π-calculus and are also meant to be used in implementing distributed systems However they are not capable of performing complex sequential computations

A representative implementation from each will be examined in detail

- From category one - JPiccola (Nierstrasz, Achermann & Kneubuehl n d)
- From category two - Pict (Pierce & Turner 2000*a*)
- And from category three - Nomadic Pict (Wojciechowski & Sewell 1999)

3.1 JPiccola

JPiccola (Nierstrasz et al n d) is a language designed for constructing applications from existing software components that are already written in another programming language All actual work is achieved via this host language, Java, and JPiccola simply provides a framework for linking these components The reasoning behind this is that existing methods of creating pluggable component architectures lack flexibility and limit designers to particular architectural styles and component models

The core of JPiccola does not provide any programming language features, only some mechanisms which facilitate the composition of components to create applications These mechanisms are related to aspects of the π-calculus, namely agents and channels Obviously since these mechanisms only allow the structuring of components and the communications between them, some method of performing actual computations is required JPiccola provides this by means of a Host programming language, the Java programming language By writing wrappers around Java code it is possible to access the functionality of the Java programming language In order to simplify the task of performing computations JPiccola provides some basic data types, such as strings, integers and Booleans, along with some basic control structures These are provided via standard JPiccola modules which perform the required wrappings

JPiccola differs from the π-calculus in terms of both syntax and semantics However, the π-calculus concept of mobility, static agents yet dynamic links between them, is present in JPiccola JPiccola is also not distributed in nature, and all communications over channels in it are asynchronous in nature JPiccolas main strength comes from its ability to perform complex sequential computations via a "host" language, namely the Java programming language

3 1 1 Forms and Services

Central to JPiccola is the concept of a form A form in JPiccola consists of a series of name-value bindings and services that allow these forms to be invoked e g

```
person =
   name = ''john doe''
   age = 31

   printName
      println ''Name '' + name

   printDetails
```

```
        print ''Name '' + name
        println '' Age '' + age

person printName()
person printDetails()
```

The above JPiccola code creates a form, *person*, which contains two name-value bindings and two services The two services, *printName* and *printDetails*, are then invoked to yield the output

```
Name john doe
Name john doe Age 31
```

Much like inheritance in object-oriented languages forms can extend other forms, thereby gaining the services and name-value bindings of other forms

```
student =
    person
    stuno = 99999999

    printDetails
        print ''Name '' + person name
        print '' Age '' + person age
        println '' ID No '' + stuno

student printName()
student printDetails()
```

The form *student* can access all the name-value bindings created in the *person* form, and it can also access all the services that *person* provides Name-value bindings must be explicitly referenced while services do not Services can also be overridden, e g the *printDetails* service in the *student* form

3 1 2 Concurrency and Interaction

JPiccola is heavily influenced by the π-calculus and as such interaction and concurrency in JPiccola is achieved in a manner similar to that of the π-calculus

Concurrency in JPiccola is achieved by invoking *run* on a service, this causes the service in question to be executed in parallel to the rest of the invoking entity

e g

```
run (do student printDetails())
run (do person printDetails())
```

This code sets the two services, *student printDetails* and *person printDetails*, running concurrently to the invoking service The result of the above will be the printing of both sets of details to the screen in an arbitrary order Obviously services that can be more complex are possible to be invoked in a concurrent

fashion, and no bounds are placed on how deep nested concurrent invocations can be

When dealing with multiple agents running concurrently the issue arises of communication and interaction between these agents JPiccola once again turns to the π-calculus for the solution The concept of a channel was introduced to JPiccola, and agents can communicate with each other over these Channel communications in JPiccola are done in an asynchronous fashion Any attempts to send information on a channel are deemed to have succeeded instantly and do not block, while reads are done in a blocking fashion JPiccola channels allow only the *communication* of forms, but since everything in JPiccola is a form this is not a problem

```
   e g
c = newChannel()

run (do  println c receive())

run (do  c send(''hey from over here''))
```

3 1 3 The Host language

JPiccola has no built in means of performing computations, rather it delegates all computations to a host language, Java Everything that is possible in the Java programming language can be achieved in JPiccola by means of wrappers Wrappers for the most common data-types are supplied with JPiccola, and it is via these wrappers that the "built-in" types like numbers, strings and Booleans are supplied In order to access other types of Java objects it is necessary to use the *Host class* service This service returns a form and all functions available to the Java object are available as services that can be invoked on the form

```
   e g
dig = Host class(''java security MessageDigest'') getInstance(''SHA1'')
dig update(''thisisastring'' getBytes())
res = dig digest()
```

By combining this method of utilising the Java programming language along with JPiccolas built in control structures and array access methods, the full expressivity of the Java programming language can be accessed and used

3 1 4 JPiccola Summary

For some time there has been some concern with regard to the manner in which the Java programming language handles communications and interactions between concurrently executing threads JPiccola manages to overcome this problem with the addition of channels This addition ensures that all interaction

between concurrent agents is achieved in a transparent fashion As JPiccola also allows the full computational power of the Java programming language to be harnessed it could be argued that it is possible to achieve more in it than in pure Java This ability to include Java code in JPiccola programs, albeit in a round about manner, is one of the distinguishing features of JPiccola and it is this capability that ensures that JPiccola is actually of some use to programmers

While JPiccola allows access to the capabilities of the Java programming language, and while it also includes certain aspects of the π-calculus, it is visually similar to neither This, combined with the minimal influence of the π-calculus seems to have had on it, means that JPiccola's value to the formal methods community, and more specifically to those concerned with implementing π-calculus specifications, is rather limited It could also be argued that for what JPiccola does, that it is overly complex

However the most significant drawback of JPiccola is that it does not cater for distributed systems JPiccola is primarily designed for users implementing stand-alone applications, which may perhaps be concurrent in nature The primary concern of the π-calculus is the specification of protocols and interaction between distributed entities and as such the π-calculus is of limited use with regard to systems that will only be executed on a single machine The non-distributed nature of JPiccola significantly reduces its attraction to those involved in the specification and implementation of distributed systems

Classification of JPiccola

Syntax	Not similar to π-calculus
Semantics	Not similar to π-calculus
Mobility	π-calculus mobility present
Channels	Asynchronous
Distributed	No
Sequential computations	via "host" language, expressive and powerful

)

3.2 Pict

Pict was developed by B Pierce and D Turner in the late 1990's(Pierce & Turner 2000*b*, Pierce 1997) as an experiment to see what a language based on the π-calculus would look like The idea behind Pict was to design and implement a high level language purely in terms of the π-calculus and as such Pict is intended to be to the π-calculus what Lisp (Seibel 2005), ML (Paulson 1996) or Haskell (Thompson 1999) are to the λ-calculus (Thompson 1999) As would be expected goals, Pict is very similar, syntactically and semantically to the π-calculus Another result of this intention is that the traditional π-calculus form of mobility is present in Pict However this intention is also the source of one of the major problems with Pict - its inability to perform complex sequential computations Sequential computations in Pict are achieved via Pict's own notation, which is, in effect, an extension to the π-calculus Finally, like JPiccola, Pict is not distributed in nature and channel communications are asynchronous

Code written in Pict is visually very similar to π-calculus specifications and concepts present in the π-calculus are, for the most part, present in Pict This allows nearly everything possible in the π-calculus to be done in Pict Pict is a completely self-contained language which allows everything, communications and computations, to be achieved in its own notation This unique notation is, simultaneously, one of the strengths and weaknesses of Pict

Pict code is compiled into C code, and from C into executables Once compiled, these strongly typed programs run in a uniprocessor *NIX environment like any other traditional C programs

3 2 1 Processes and channels

In Pict, much like in the π-calculus, everything is arranged in terms of processes Also in a similar manner to the π-calculus is the construction of processes Processes are made up of a number of actions and a number of sub-processes, and the arrangement of these sub-processes and actions is done in a fashion similar to that of the π-calculus Pict also allows the concurrent execution of an arbitrary number of processes

Example

```
run(print''hello'' | print''world'')
```

The Pict version of the standard *Hello world* example program is a process which involves the parallel execution of two sub-processes - one that prints "hello" and one that prints "world" However, as would be expected of two processes executing in parallel without any form of interaction, the ordering of the output is non-determimstic - one time it may say "hello world", the next it may say "world hello" As can be seen processes in Pict are invoked by enclosing

them in parentheses and separating them with the parallel operator, "|", after having prefixed the entire expression with the keyword *run* Even at this early stage it is quite obvious of the syntactic and semantic similarities between Pict and the π-calculus

Processes in Pict have, like their π-calculus equivalents, only one method of inter-process interaction available to them - channels Channels in Pict are notably different to channels in the conventional π-calculus as Pict channels are both asynchronous in nature and strongly typed Each channel may be used to send and receive values of only one type, and this restriction removes the possibility of implementing some very reasonable and useful programs This restriction does reduce the expressiveness of Pict but it is claimed that not restricting it would have resulted in major implementation issues(Pierce & Turner 2000*b*, Pierce 1997) Asynchronous channels still allow the communication of data between processes, but have a slightly reduced capacity for allowing processes to synchronise their execution

Example

```
new x  []
run (x?[] = print!''Hello world'' | x![])
```

There is another Pict version of the *Hello World* example program This version also consists of two sub-processes running concurrently, however in this case one of the sub-processes prints "Hello World!" after receiving data on a channel, while the other sub-processes simply invokes the other by outputting on the shared channel

In Pict there is a distinction between names and channels, and creating a new channel requires an explicit declaration of its identifier and its type, i e the type of data that will be transmitted on it In the above example the channel x is created and it is given a type of [], this means that the channel will not actually carry any data, but rather will only be used to "invoke" the printing process

Example

```
new x ^String
run (x?y = print!y | x!''Hello World'')
```

The previous *Hello world* example logically leads onto this one In this version not only is the sub-process that prints the message invoked by another sub-process but the data that it is to print is also sent to it by the invoking channel action In the above example the channel x is created with the capability of transmitting data of type *String* and only data of type string, any attempt to do otherwise will cause a compilation error

Example.

```
new x ^String
new q ^^String
run (q!x | q?y = y!''hello world'' | x?a = print!a)
```

The communication of channels from one process to another is also allowed in Pict, this allows processes to learn of new channels and introduces the concept of mobility to Pict In this example a channel is received by one of the sub-processes and then this channel is used to communicate with a previously unavailable process The result of this example is the same as the others, the printing of "hello world" to the screen

3 2 2 Built in types

A number of built-in types are included in Pict, including the most commonly used types, strings, integers and Booleans The usual operations can be performed on these types, but these operations are performed in a unique Pict fashion

Integers

Integers in Pict are considered to be processes that are "located" at specific channels The values of these numbers can be gained by querying these processes over the channels

Example [1]

```
new r ^Int
run (+![2 3 (rchan r)] | r?x = printi!x)
```

In the above example the numbers *2* and *3* are to be thought of as processes located at the channels *2* and *3*, while *r* is thought of as channel to a process which sums its arguments and then makes the result, in this case *5* available on the channel *r* Other operations, such as multiplication, division, subtraction, etc, etc, are available using the same *op*![*abw*] notation Integers in Pict are printed via the *printi* command, rather than the *print* command which is only used with strings

Strings

Normal String operations are possible in Pict, these include

Concatenation

```
new x ^String
run( x!''hello'' | x!'' world'' | x?a = x?b = print!( b a))
```

[1] The use of the keyword *rchan* is not directly related to Integers but rather to process definitions Channels used with defined processes can sometimes be required to be of a special type, *rchan* forces a normal channel to act as one of these special channels

In this concatenation example, the final sub-process reads two strings in over the channel x, and then prints the result of joining these two strings together, "hello world"

Sub-Stringing

```
new x ^String
run( x!''hello world test'' | x?a = print!(string sub a 0 11))
```

The ability to obtain sub-strings from strings is an operation that is vital to any implementation of strings, Pict's implementation of Strings does provide this capability via the *string sub* command, which takes the string and the start and end index of the sub-string to be gotten from the string in question

Sub-String tests

Sometimes when dealing with strings is it necessary to test if a string contains a certain sub-string, Pict implementation of Strings allows this by using the *string in* command

```
new x ^String
run( x!''hello world test'' | x?a = if( string in ''hello'' a)
        then print!''There'' else print!''Not there'')
```

This example also introduces one of the control structures present in Pict, the if statement If statements in Pict can include an arbitrary number of else-if options This additional control structure compliments those already present in the π-calculus

Other types

These examples are by no means intended to be an exhaustive explanation of the built-in types in Pict, rather a brief introductory glance at how certain operations are performed in Pict It is intended to give an impression of how tasks are completed in Pict, rather than detail all the available operations and primitives as Pict provides a large number of built-in types, and allows a vast array of operations to be performed on these types

Other built-in types include

- Booleans
- Lists
- Characters
- Floats
- Bytes

- Queues
- Arrays
- GUI related types

3 2 3 Process definitions

Pict allows the explicit definition of processes, much like a function call in an object oriented language these definitions reduce the amount of code in programs and make it much more convenient to write larger programs

Example

```
def whatIsMax c ^Int = c?a = c?b = if(>> a b) then c!a else c!b

new f ^Int
run( whatIsMax!f | f!3 | f!4 | f?a = printi!a)
```

Process definitions save both time and effort when writing Pict programs and of course should be used wherever there is duplication of code

3 2 4 Pict Summary

Pict is an invaluable experiment in the investigation into languages based on the π-calculus It looks and behaves in a very similar manner to the π-calculus and the essential concepts of the π-calculus, such as processes, channels and names, are present in Pict, albeit in a slightly altered form

It is these slightly altered forms that somewhat reduce the usefulness of Pict, the strongly typed, asynchronous channels restrict the set of implementable systems, and greatly complicate some of the remaining possible systems

One of the key features of Pict is that everything it does, it does via its own notation - no host languages or the like are used All communications and computations are achieved via this unique notation However there are negative aspects to this approach The notation used is somewhat complex and counter-intuitive in places and is also rather limited in what can be achieved, basic computations and GUI related programs are possible but some implementations would be far beyond the capabilities of Pict, e g a complex cryptographic computation

Pict has substantial theory and documentation behind it, which makes it not only a good introduction to programming languages based on the π-calculus, but also to the π-calculus itself However the lack of support for distributed systems and communications in Pict means that Pict is not the most useful language to use when implementing concurrent and distributed π-calculus specified systems

Classification of Pict

Syntax	Comparable to the π-calculus
Semantics	Comparable to π-calculus
Mobility	π-calculus mobility present
Channels	Asynchronous
Distributed	No
Sequential computations	Limited

3.3 Nomadic Pict

Nomadic Pict (Wojciechowski & Sewell 1999) was developed by Wojciechowski, Sewell and Pierce in 2000 It is intended to be a programming language that is based on the π-calculus that allows the concurrent and distributed execution of systems that are implemented in it Nomadic Pict is built on the programming language Pict and Pict primitives are used to express computations within Nomadic Pict agents

Nomadic Pict uses agents as the building blocks for systems Agents can be viewed as collections of communications and computations required to achieve specific goals These agents are mobile and can "migrate" from one host machine to another

As Pict is used for all computations in Nomadic Pict, the only aspect of Nomadic Pict not previously covered is the communication of agents, both distributed and non-distributed The additions and alterations to Nomadic Pict can be grouped into two main sections Agents, Sites & Migration, and Channel Actions

Although Nomadic Pict is based on Pict, the syntax and semantics have been so drastically altered that the syntax and semantics of Nomadic Pict are no longer similar to those of the π-calculus Furthermore, changes to how the asynchronous channels of Pict operate, the traditional π-calculus form of mobility is absent in Nomadic Pict However, a form of mobility is present, the form of mobility that arises from the ability of agents to re-locate from one point of execution to another Nomadic Pict, being based on Pict, inherits the problems associated with Pict's approach to sequential computations - it cannot perform complex sequential computations However, unlike Pict, Nomadic Pict is distributed in nature and agents of a system can be executed in an arbitrary, concurrent, and distributed manner In short, Nomadic Pict draws more from other process calculi than it does from the π-calculus, namely the Ambient calculus (Cardelli & Gordon 1998)

3 3 1 Agents, Sites and Migration

Three concepts not present in standard Pict are introduced into Nomadic Pict in order to facilitate distribution communications - agents, sites and migration An *agent* in Nomadic Pict is a unit of executing code and each unit has a distinct name, which refers to a body comprised Nomadic Pict/standard Pict actions Since systems implemented in Nomadic Pict will be distributed amongst many machines in an arbitrary fashion, some method of representing the possible locations of aspects of the systems is required, each possible location is called a *site* Communication between different agents residing in different sites is possible though not in the π-calculus sense, however a different form of mobility is also available in Nomadic Pict Agents in Nomadic Pict have the capability of *migrating* from one site to another, they can change the machine on which they are executed

Example

```
program param  [Site Site] =
(
val siteOne = (get_site 0)
val siteTwo = (get_site 1)

new answer ^String

agent homeBody =
(
        agent deserter =
        (
          migrate to siteTwo
          (print'''SiteTwo up and running''|
          <one@siteOne>answer'''hey from siteTwo'')
        )
(print'''SiteOne up and running''
|answer?l = print'l)
)
in ())
```

This example demonstrates the majority of the additions to Pict that comprise Nomadic Pict, as sites, agents, migration and one form of inter-agent channel communications are all covered in it The above system is intended to be run on two separate sites, or Nomadic Pict virtual machines The system is started from one site and the agent *homeBody* remains on this site where it prints a message and will eventually print another message once it has been

received from the agent *deserter* The *deserter* agent migrates to the second site where it prints its message to the screen and then transmits a message back to the *homeBody* agent that is still running on the first site

A site in Nomadic Pict represents a specific instance of the Nomadic Pict virtual machine These various instances may be running on the same machine, or on many distributed machines, the topology of the system does not affect its execution Information on the location of sites is gathered from a configuration file, and this information is used to ensure that agents migrate to the correct machines

Agents in Nomadic Pict are assigned unique names The combination of an agent name and a site name makes up the complete identifier of an agent, and it is this complete name that is used when agents are communicating It is possible for multiple instances of the same agents to run in a system, both on the same site and on different sites For example, in the above system the complete identifiers of the two agents are *homeBody@siteOne* and *deserter@siteTwo*

3 3 2 Channel Actions

Channel Output

Complete identifiers are important in Nomadic Pict because of the manner in which it performs inter-agent communications In the $\pi - calculus$, and indeed in Pict, there is only one way in which to transmit data on channels, however in Nomadic Pict there are five ways in which this can be achieved

1	$x!y$	Behaves as the standard Pict send
2	$iflocal\ < a > x!y\ then\ P\ else\ Q$	Conditional transmission of y on x
3	$< a > x!y$	y sent on x to agent a on this site
4	$< a@s > x!y$	y sent on x to agent a on site s
5	$x@a!y$	Output on x to agent a

Explanations

1 **x!y** Transmits the name y in a non-blocking manner on the channel x This form of channel output does not work for inter-agent communications, but rather only for communications between processes in the same agent

2 **iflocal < a > x!y then P else Q** If the agent a resides on the same site as the agent that is attempting to transmit the name y on x, the transmission occurs successfully and then the process P is started, if the agent a is not on the same site then no communication occurs and the processes Q is started

3 **< a > x!y** If the agent a is on the same site as the agent that is attempting to transmit the name y on x then the action succeeds, if it is not on the same site then it fails silently, i e blocks

4 **< a@s > x!y** If the agent a is on the site s then the output action succeeds, if the agent a is not on that site the output actions fails and nothing happens

5 **x@a!y** This particular type of output action differs from all the others in that it is not an implemented low-level primitive in Nomadic Pict, but rather it is a high-level construct that requires a specific implementation of it to be included in a system that attempted to perform location-independent output The creators of Nomadic Pict have included two such implementations that can be used

Consider the Nomadic Pict example give above, one occurrence of an inter-agent communication action occurs, and the line

```
<one@siteOne>answer ! ' ' Hey from siteTwo ' '
```

could have been replaced with one of four other possibilities The following illustrates the alternative possibilities and highlights the effects of using them to replace this line

- **answer!"Hey from siteTwo"**

  This form of output is only suitable for inter-agent communications and as such this attempt at output fails

- **iflocal < homeBody > answer!"Hey from siteTwo" then () else ()**

  While this form of output is suitable for inter-agent communications, it is only suitable for communications between agents that reside on the same host, however the failure of the output action is not a complete failure as this failure is detected and results in an alternative branch of execution being pursued

- **< homeBody > answer!"Hey from siteTwo"**

  This form of output is suitable for intra-agent communications, it is only suitable for communications between agents that reside on the same host Should the intended recipient not reside on the same host, this action fails silently

- **answer@homeBody!"Hey from siteTwo"**

  This is a location independent output action It is intended to deliver the message to the agent in question regardless of the site on which the sending agent is located

Channel Input

While there are a number of ways in which data can be sent on a channel in Nomadic Pict, there are only two ways in which data can be read from a channel One way to do so is in the same manner in which channel input actions are performed in standard Pict The other method for reading from channels involves the possibility of a read timing out A process waits for a predetermined amount of time for data to be available on a specific channel and if no such data becomes available then alternative actions are performed

```
wait x?y=print!``Value received''
        timeout 100 -> print!``No value received''
```

In the above fragment of Nomadic Pict code, the process will wait for 100 seconds for a value to be read on the channel x If a name is read before the time expires then the message "Value received" is printed, if none is received then the alternative course of action is taken and "No value received" is printed

3 3 3 Nomadic Pict Summary

Nomadic Pict is an extension of Pict and, like Pict, it is very similar to the π-calculus in terms of intended semantics Since it is an extension of Pict it also retains all the capabilities of Pict with regard to performing sequential computations However Nomadic Pict has a distinct advantage over Pict in that it is distributed in nature Nomadic Pict is a completely self-contained distributed programming language that is based, loosely, on a derivative of the π-calculus and as such is a valuable tool for those concerned with formal methods

However as Nomadic Pict is so closely bound to Pict it also retains a lot of the difficulties associated with Pict The strongly typed asynchronous channels still discount a lot of useful systems that, while possible in the π-calculus, are not possible in Nomadic Pict The limited number of primitives, and possible operations, on these primitives also limits what can be achieved in this language

Nomadic Picts unique approach to channels, *distribution of agents*, and inter-agent communication also raises some questions An agent wishing to communicate with another arbitrary agent must not only know a channel that is also known to the other agent, it must also have explicit knowledge of the other agent, and implicit knowledge of its location These extra restrictions on inter-agent communications drastically limit the usefulness of this language

Classification of Nomadic Pict

Syntax	Not similar to the π-calculus
Semantics	Not similar to the π-calculus
Mobility	π-calculus mobility absent, alternative form present
Channels	Asynchronous
Distributed	yes
Sequential computations	Limited

3.4 Summary

While nearly all implementations of a process calculi are valuable to those wishing to learn more about process calculi and formal methods, the usefulness of a lot of these implementations to those concerned with actually implementing distributed systems is rather limited

The major limiting factor of most implementations is their non-distributed nature, most of the implementations are designed so that systems written in them will run on only one machine In the world of security protocols this approach is almost useless as security protocols are only required for the transmission of data between multiple machines

Yet another significant limiting attribute of the majority of implementations is the poor expressive capabilities of some of the languages with regard to sequential computations The bulk of the languages choose to perform all sequential computations through primitives and operations of their design, however these languages tend not to have the number, or diversity, of primitives or operations, required to implement complex and computational intensive systems

Combined these limiting factors results in existing implementations of languages based on the π-calculus being of little real-world use, what is required is a language that is both based on the π-calculus and that is also highly expressive

3 5 Conclusions

The language presented in the following sections is one that is syntactically and semantically very similar to the π-calculus It is one in which the traditional π-calculus concept of mobility is present, and is one in which all communications are synchronous in nature It supports the distribution of systems written in it, and it allows the arbitrary deployment of distributed systems It also provides a mechanism for performing complex sequential computations in a manner reconcilable with the π-calculus

Once the language design(chapter four) and the language implementation(chapter five) have been outlined and explained ϖ will be rated against the classification criteria described in this chapter

Desired Classification of ϖ

Syntax	Very similar to the π-calculus
Semantics	Very similar to the π-calculus
Mobility	π-calculus mobility present
Channels	Synchronous
Distributed	yes
Sequential computations	Powerful and expressive

Chapter 4

ϖ - The language

Distributed systems are becoming increasingly commonplace The use of formal notations and their associated formal methods, such as the π-calculus, and its derivatives, in ensuring that these distributed systems are in fact secure is also becoming more routine and established Yet no programming language that is suitably usable, expressive, distribution oriented and incorporates the π-calculus notion of mobility, exists and as such there is a niche for such a programming language that is based on the π-calculus

Such a language would have to satisfy two criteria The first being that it should have a close relationship with the π-calculus The second being that it should be capable of implementing distributed systems in a simple and transparent manner In order to fulfil these two end goals a series of sub-goals must be satisfied

The typical distributed system requires that a number of complex operations be performed These complex operations, such as the generation of keys, encryption and decryption of data, hashing, creating and verifying signatures, transmission of data, etc, are complex and computationally intensive Various programming languages contain a number of cryptographic and networking primitives and operations that greatly simplify the programming of these distributed systems While the π-calculus is computationally complete and it is theoretically possible to express cryptographic operations in it, it would be impractical to do so given the size and number of π-calculus statements that would be required The addition of mechanisms for performing these cryptographic operations to the communications capabilities of the π-calculus would yield the desired result - a powerful and expressive programming language based on the π-calculus suitable for use in implementing distributed systems

So in order for a language to be "capable of implementing distributed systems in a simple and transparent manner", it must provide an apparatus for handling

the distributed nature of the target systems while also supplying mechanisms for performing complex operations

It is also desired that the programming language is closely modelled on the π-calculus, that it looks and acts in a manner similar to the π-calculus, while also being computationally more usable This close modelling may allow the application of the formal techniques associated with the π-calculus in order to verify the correctness of protocols In other words, despite any additions required in order for the implementation of distributed systems, the syntax and semantics of the programming language must be as similar as possible to those of the π-calculus Further still, this similarity must be attained in a manner which will allow the language to be closely coupled with extensions of the π-calculus, such as the Spi-calculus, and not just the core calculus itself

Central to achieving the desired syntactic and semantic similarities between this language and the π-calculus is the integration of computations into the communications aspect of systems This integration is made possible by the dual nature of data items in this programming language In one form these data items exist as names, and in the communications aspect they can be used in the transmission of names either as the transmitter, or that which is being transmitted While in their other form they are simply objects in an object-oriented programming language An object created in the computational code and be brought into the communications code and transformed into a name, and likewise, a name created in the communications code can be pushed into a computation and transformed into an object This dual nature of data items is a pivotal concept in this language, and as such is vital to the understanding of its syntax and semantics

At first glance the completion of these two goals appears to be somewhat mutually exclusive The presence of mechanisms for performing complex computations would seem to be at odds with maintaining the syntax and semantics of the π-calculus It is felt that the language, ϖ, should achieve both these goals An attempt was made to satisfy these goals via the language definition and the implementation of ϖ The design and structure of the programming language ensured the similarities between it and the π-calculus were present, and also solved the problem of reconciling a mechanism for performing computations with the syntax and semantics of the π-calculus The issues related to the distributed nature of the programming language were resolved via the actual implementation of the programming language, as were some aspects of integrating sequential computations into the syntax and semantics of the π-calculus Those requirements that were satisfied via the first approach are covered in this chapter, while those that were solved by the second method are covered in a later chapter

4 1 What is ϖ?

The π-calculus is computationally complete, that is, it is theoretically possible to perform any computation using only the existing syntax and semantics of the π-calculus However there is a massive difference between computability and usability, and while it is *theoretically* possible to, for example, compute the result of encrypting some data using existing π-calculus features it most certainly isn't realistic to do so

ϖ (var-pi) is the result of an attempt to facilitate the performing of sequential computations, simply and transparently, in a π-calculus influenced framework It is hoped that ϖ could be viewed as the π-calculus with computations Or given that the set of sequential computations available in the Spi-calculus is a subset of those available in ϖ, it is also hoped that it could be viewed as the Spi-calculus with a broader range of sequential computing capabilities

4 1 1 Abstract Syntax and Semantics of ϖ

A goal central to the success of ϖ is the concept of a close coupling between the specification language, the π-calculus, and the implementation language, ϖ itself Obviously in order to ensure that ϖ looks and acts in a manner akin to the π-calculus it must have a syntax and semantics that are similar to those of the π-calculus However, as the π-calculus is a specification tool and ϖ is a programming language, it is inevitable that the actual syntax and semantics of ϖ will be more complex than that of the π-calculus - brackets, commas, braces, colons and the like all become, unfortunately, necessary As such at this stage the abstract syntax of ϖ will be used in any comparisons made with the π-calculus The concrete syntax of ϖ will be given later

This abstract syntax can then be used for the initial analysis and comparison between the π-calculus and ϖ After inspecting the syntax and semantics of ϖ it should become clear that a ϖ system will look and behave similarly to its original π-calculus specification

Abstract syntax

In the following description of the ϖ syntax we let m, n range over names, x,y range over variables, and let f, g range over the set of valid function identifiers

Terms

L,M,N =	n	name
	x	variable

Output Action

α =	$\overline{N} < M >$	Output action

Input Action

β =	$N(M)$	Input action

Processes

P,Q =	α P	Input prefix
	β P	Output prefix
	τ P	Unobservable prefix
	!P	Replication
	(νn)P	Restriction of channels
	$(f(L_1 \quad L_m)(x_1 \quad x_o))$P	Restriction of non-channels i e performing a sequential computation
	P | Q	Composition
	α P + α Q	Guarded sum
	$[N == N']$P + Q	Match
	0	Null process

Explanation

1 **Input prefix** The relevant input action is performed, and the process continues as P with any necessary substitutions being made in P

2 **Output prefix** The relevant output action is performed, and the process continues as P

3 **Unobservable prefix** An unobservable interaction occurs and the process continues as P

4 **Replication** The process $!P$ is equivalent to $P \mid !P$ In other words $!P$ behaves as an arbitrary number of instances of P all executing in parallel

to one another

5 **Restriction of channels** Create a new name, n, of type channel and binds it in P

6 **Restriction of non-channels** Creation of names of the type non-channel is achieved by the execution of sequential computations In this form of restriction f is a computation It takes a series of input terms L_1, L_m, which it pushes down into the computation in question which yields a series of names These produced names then replace the series of input variables x_1, ,x_o in P

7 **Parallel Composition** Both the processes P and Q are executed concurrently These processes can interact with each other and with other processes

8 **Guarded sum** Interaction can happen with *either* P or Q but not both Which process is started depends entirely on which input action occurs first

9 **Match** If the N is equal to N' then the process behaves as P, otherwise the next option in the Match statement is processed, this may be another match condition or the default process, Q

10 **Null process** The empty process, it cannot do anything

Structural Congruence

As can be imagined it is very possible to construct two processes that behave in an identical fashion but yet are syntactically dissimilar A structural congruence is used to equate these processes that intuitively represent the same process Two processes P and Q are said to be structurally congruent, $\equiv$, if $P \equiv Q$ can be inferred from the axioms listed below, and by alpha conversion These axioms allow manipulation of term structure and are not reliant on the semantics of the language

SC-SUM-ASSOC	$P_1 + (P_2 + P_3) \equiv (P_1 + P_2) + P_3$
SC-SUM-COM	$P_1 + P_2 \equiv P_3 + P_1$
SC-SUM-INACT	$P + 0 \equiv P$
SC-COM-ASSOC	$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$
SC-COM-COMM	$P_1 \mid P_2 \equiv P_2 \mid P_1$
SC-COM-INACT	$P \mid 0 \equiv P$
SC-REP	$!P \equiv P \mid !P$
SC-RES	$(\nu m)(\nu n)P \equiv (\nu n)(\nu m)P$
SC-RES-INACT	$(\nu n)0 \equiv 0$
SC-RES-COMP	$(\nu n)(P_1 \mid P_2) \equiv P_1 \mid (\nu n)\ P_2$, if n $\notin$ fn(P_1)
SC-MATCH	$[n == n]P \equiv P$

Discussion

A brief visual comparison between the abstract syntax of ϖ, given above, and the syntax of the π-calculus, given in section 2 1 2, clearly shows the similarities between the two syntaxes By and large the syntax of ϖ is almost identical to that of the π-calculus, bar the addition of variables, and could even be mistaken for the syntax of a variant of the traditional synchronous π-calculus, in particular the Spi-calculus, rather than an implementation of it

Systems are still organised as a series of processes running in parallel, processes are still constructed from a series of valid actions, and processes can still be replicated The actions available to be performed by a process remain the same, (names can be sent and received on channels and internal reaction can occur in a process), as do the methods for invoking other processes, (choices can be made between processes, processes can be executed concurrently and processes can be invoked only after an equality test is satisfied)

For reasons that will be outlined at a later stage it was necessary to break names into two categories - channels and non-channels As such names in ϖ are either of type *channel* or of type *data* The difference being that names of type *data* do not have the capability to communicate other names, while names of type *channel* do Due to the complications introduced by the ability to communicate names at run-time it was felt that the type checking would be more suited to run-time rather than compile time

As can be seen the syntax of ϖ differs from that of the π-calculus in only a few places The first, and most significant, being the introduction of a rudimentary typing system and the addition of a second form of restriction The next, and less significant, difference is the imposition of a constraint on summations in ϖ In ϖ all summations must be guarded summations, and furthermore these guards must always be input actions, the reasoning behind the restrictions on

summations is explained later in this chapter The final difference between the syntax of ϖ and that of the π-calculus is the addition of the notion of a "default" process in a match statement That is, should all the match conditions in a match statement fail then there is a process present that will be invoked in this case The impact of these new elements will be discussed in section 4 1 1

However, *this similarity in syntax* only shows that the two *look* the same, in order to demonstrate that they *act* in the same way the semantics of ϖ must be examined

Operational Semantics

[Struct] $$\frac{P' \equiv P,\ P \xrightarrow{a} Q,\ Q \equiv Q'}{P' \xrightarrow{a} Q'}$$

[Prefix] $$\frac{}{a\,P \xrightarrow{a} P}$$

[Par] $$\frac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q}$$

[Com] $$\frac{P \xrightarrow{x(y)} P',\ Q \xrightarrow{\overline{x}<n>} Q'}{P \mid Q \xrightarrow{\overline{x}<n>} (P'\{n/y\} \mid Q')}$$ if x is of type channel

[Match 1] $$\frac{}{[x==x]P \longrightarrow P}$$

[Match 2] $$\frac{x \neq y}{[x==y]P+Q \longrightarrow Q}$$

[Res1] $$\frac{P \xrightarrow{a} P',\ x \notin a}{(\mathbf{v}x)P \xrightarrow{a} (\mathbf{v}x)P'}$$

[Res2] $$\frac{}{(\mathbf{f}(L_1 \ \ L_o)(x_1 \ \ x_m))P \xrightarrow{\tau} (\mathbf{v}n_1 \ \ n_m)P'\{\frac{n_1 \ \ n_m}{x_1 \ \ x_m}\}}$$

[Sum] $$\frac{\alpha_1 P \xrightarrow{\alpha_1} P}{\alpha_1 P + \alpha_2 Q \xrightarrow{\alpha_1} P}$$

Explanations

1 [**Struct**] If the occurrence of an action causes the process P to reduce to

the process Q, then a process that is structurally congruent to P can be reduced to a process that is structurally congruent to Q on the occurrence of the same action

2 [**Prefix**] A process that is prefixed by an action reduces to that process after the occurrence of the specific action

3 [**Par**] If a process, P, can reduce to another process, P', after the occurrence of an action then P will reduce to P' regardless of what processes are running concurrent to it when that action, a, occurs

4 [**Com**] If a process P reduces to P' on an input action on a specific name, which is of type *channel* and if the process Q reduces to Q' on an output action on that same name then P in parallel to Q will reduce to P' in parallel to Q' after an unobservable action occurs

5 [**Match1**] A process prefixed by a match statement will reduce to the process if the names are the same

6 [**Match2**] A process prefixed by a match statement in parallel with another process will reduce to the other process if the names are not the same

7 [**Res1**] If P reduces to the process P' on an action, and the name x is not involved in this action, then the reduction will only occur if the name x is restricted in both processes

8 [**Res2**] f is a computation A computation uses a series of input terms to create a specified number of names Once created these names replaces all occurrences of the indicated variables in the remainder of the process

9 [**Sum**] If a process, P, can reduce to another process, P', after the occurrence of an input action then the sum of P and any other processes can reduce to P' on the occurrence of that input action

If a comparison is made between the semantics of ϖ outlined above and the semantics of the π-calculus given in chapter two, it becomes immediately obvious that these two sets of semantic rules are similar The differences between them arise from the constraints placed on summations, the insistence that names used to transmit other names are of type *channel*, and the addition of an additional rule for the restriction of names of type *non-channel* The impact of these differences will also de discussed in section 4 1 1

In fact the semantics of ϖ are so close to those of the traditional π-calculus that they could easily be mistaken from the semantics of a variant of the π-calculus rather than those of a programming language based on it As desired

the addition of mechanisms for performing complex sequential computations has had minimal impact in the syntax and semantics of ϖ

This similarity of semantics is much more important than any syntactic similarities as it is more important that ϖ and the π-calculus act the same than they look the same

Syntactic and semantic differences

Considerable efforts were made to ensure that ϖ and the π-calculus look and behave in a comparable fashion However divergences between the two were inescapable and the two do in fact differ on three issues

Difference One - Summation

Some variants of the π-calculus permit summations of an unguarded nature to occur, unguarded meaning that processes occurring in summation need not be prefixed by an action, e g $P + Q$ However more variants of the π-calculus use guarded sums instead of unguarded sums as the theory behind the π-calculus is simplified somewhat by this decision(Parrow 2001) As the choice as to which process is started in an unguarded summation is a non-deterministic choice, unguarded summations would be of little use in a real world programming language where totally random actions of this kind are rarely desired, and often discouraged As such the constraint that all summations must be guarded was imposed on summations in ϖ

However guarded summations are not without their implementation issues Guarded sums are often said to be "unrealistic from an implementation perspective" (Parrow 2001), as the decision as to which guard in a summation occurs can prove to be a non-trivial problem The problem results from attempting to match input actions to output actions when both types of action are conditional This, combined with multiple summations in parallel, results in the general form of guarded summations not being a realistic operation from an implementation point of view In particular in a distributed environment, if summations in which mixed guards are allowed to occur in parallel to each other, it is possible, that no action will occur No realistic method of implementing a mixed guarded summation that is stable, reliable, and whose behaviour was guaranteed, exists Consequentially it was decided that the only valid guards for statements in a summation in ϖ would be input prefixes, in other words output actions are always guaranteed to occur, while the completion of input actions can be conditional

A quick comparison between the relevant aspects of both syntaxes and semantics reveals that this difference is not a major one, but rather merely a

minor restriction placed on what constitutes a valid summation in ϖ By placing this restriction on the guards of a summation this impossible problem of mixed guards is avoided with the minimum loss of expressiveness In fact nearly all systems involving mixed guards in a summation can be re-written to include only input guards The exceptions arising when both "ends" of a communication are both in summations In these scenarios the original behaviour can not be approximated using only input guards However this loss of expressivity is not an overly significant one and only occurs in systems such as $(\bar{a}x\ P\ +\ b(y)\ Q)\ |\ (a(z)\ R\ +\ \bar{b}w\ S)$

Difference Two - Typing

The major difference between the syntax and, more importantly, the semantics of the π-calculus and of ϖ is the introduction of a rudimentary typing system to ϖ

The ϖ typing system divides all names in a ϖ system into two types - those names that have the capability of acting as channels, and the names that do not have this capability While only names that have the capability of acting as channels can be used to communicate other names, names of both types can be communicated on channels This typing system was imposed solely for implementation reasons, and the imposition of it greatly simplified the implementation of ϖ This typing system does however have an effect on the flexibility of ϖ Greater attention must be paid to the use of channels than in the π-calculus in order to avoid run-time errors

With regard to the semantics of ϖ, the typing system only affects two of the semantic rules - the new rule that governs the restriction of fresh non-channel names and the one related to the interaction of concurrent processes over a specific channel The former rule was required to be added to the set of semantic rules in order that fresh names of type non-channel could be created In the latter the changes to the corresponding π-calculus rule are even more minor - it now insists that all names used for communicating other names between processes be of the type *channel*, i e that they have the capability of acting as a channel, a simple and obvious requirement

This difference between the syntax and semantics of the π-calculus and ϖ, the addition of a typing system, does not result in ϖ and the π-calculus being irreconcilable, far from it in fact as any π-calculus specification can be rewritten in ϖ if one uses only ϖ names of type channel

Difference Three - Inline Code

The final syntactic and semantic difference between ϖ and the π-calculus is ϖs ability to "inline code" into the communications aspect of ϖ processes This "inlining of code" refers to the capability of ϖ processes to perform complex sequential computations in a simple, transparent and intuitive fashion, i e via the restriction of non-channels operator This capability has such a significant impact on ϖ that it will be covered in great detail in section 4 2 1

4 1 2 Concrete Syntax

One of the primary uses of an abstract syntax is to allow properties of a language, or a program written in that language, to be reasoned about In this case less is indeed more and the less detail that appears in an abstract syntax the simpler the reasoning process is However this high level description of the form of a language is not a sufficient blueprint to use in implementing both the language and programs written in that language As such a more fine grained syntax is required This syntax is known as a concrete syntax Generally speaking a concrete syntax could be viewed as the abstract syntax with the addition of keywords, delimiters, scope boundaries, constructs for process abstraction and other real world syntactic necessities It is also common for a rule that appears in the abstract syntax to be broken down into more than one syntactic rule in the concrete syntax

As would be expected, and as can be seen below, the concrete syntactic rules of ϖ are many times more complex, and many times more numerous, than their abstract counterparts However, while this concrete syntax may be more detailed and complex than the associated abstract syntax, a simple reduction and merging process can yield the abstract syntax from these concrete syntactic rules

Syntactic Rules

System	=	[Imports] TopLevelProcess (ExplicitProcess | JavaBlock)*
Imports	=	"{" (javaPackageName ",")+
TopLevelProcess	=	"System" processId "{" (ChannelDec " ")* ProcessInvocation "}"
ExplicitProcess	=	"Process" processId "(" [list] ") "{" ProcessBody "}"
Process	=	ProcessBody | ProcessReference
ProcessBody	=	(ProcessStm)* (ProcessInvocation | ProcessChoice | MatchStart | nullProcess)
ProcessReference	=	ProcessId "(" [list] ")"
ProcessInvocation	=	"(" ["!"] Process ("|" ["!"] Process)* ")"
ProcessChoice	=	("+" "("ChannelAction ProcessInvocation")")+
MatchStart	=	("[" name "==" name "]" ProcessInvocation)+ ProcessInvocation
ProcessStm	=	(ChannelDec | JavaCode | SimpleChoice | ChannelAction) " "
ChannelDec	=	"Channel" list
JavaCode	=	"<"(code | JavaReference) ">" "(" [list] ")" "(" [list] ")"
SimpleChoice	=	list "(" name ")"
ChannelAction	=	name "(" name ")"
	=	name "<" name ">"
JavaReference	=	name | ProcessId
name	=	lowerCaseLetter (alphaNumeric)*
ProcessId	=	upperCaseLetter (alphaNumeric)*
list	=	null — name ("," name)*
nullProcess	=	0
JavaBlock	=	"Code" "(" [list] ")" "(" [list] ")" "{" code "}"
code	=	Java code enclosed in "/&" and "&/"

where

- " " indicates that the contents are a literal
- [ ] indicates that the contents are optional
- + one or more of the preceding statement
- * zero or more of the preceding statement

- | choice between statements

4.2 ϖ features

4.2.1 Sequential Computations

The absence of the promised mechanism for performing complex sequential computations may have been noticed by this stage However this perceived absence is a result of the subtle manner in which the mechanism has been added to ϖ A mechanism for performing sequential computations is present in ϖ, as whenever a name of type non-channel is created it is created as a result of a sequential computation As such the body of the second restriction operator, [RES2], generally will consist of "inline code", or a reference to a collection of such code, that will yield the necessary fresh name(s) given a, possibly empty, sequence of existing names

The minimal impact of the addition of this mechanism to ϖ is a direct consequence of the separation of the communications aspect of ϖ from the computations part of it Paradoxically complete separation of computations and communications allowed the seamless integration of them and the benefits of this complete separation of communications and computations are not limited to allowing the mechanism to be added with only inconsequential alterations to the semantics of ϖ, the benefits are, in fact, varied and far-reaching

One of the more obvious advantages of the separation of computational code from communications related code is that this separation allows the separate development of both aspects of a system Separate development of computations and communications allows them to be developed in a correct and proper manner, testing the result of a complex computation when that computation is embedded in the middle of a highly complex protocol can be problematic at best Separate development can help to ensure that not only does the protocol operate as desired but the sequential computations act as expected In effect the development of a ϖ system could be viewed as two separate development tasks, each with different goals which are achieved using two different programming languages This simplification of the development process can yield significant savings in time and effort for reasonable sized projects

Yet another benefit of developing the computational side of a system separate to the communications aspect of that system is that the development of both need not be done by the same individual(s) People completely unfamiliar with formal methods, the π-calculus, distributed systems, and indeed even of ϖ, can develop the bulk of the computational aspects of a system, leaving more experienced, and expensive, people with less work required to merge the compu-

tations and communications, thus reducing the cost of implementing systems

Complete separation of communications and computations allows the mechanism for performing these sequential computations to be added to ϖ without "polluting" the syntax and semantics of ϖ with regard to the π-calculus It is this unique approach to performing sequential computations in a language based on the π-calculus that allows the syntax and semantics of the communications aspects of ϖ to be kept as simple as possible

The sequential computations in ϖ are completed by using an "embedded language", that is fragments of the Java programming are used to perform the necessary calculations By using Java as the embedded language even the most complex sequential computations can be performed in ϖ While the Java programming language was the language chosen in this case, it would have been possible to have used any programming language in its place

Another benefit of using snippets of Java code to perform sequential computations arises from Java being such a popular and familiar language which will accelerate the ϖ learning curve

It was necessary to place certain restrictions on what can be achieved in Java to ensure that the π-calculus model is not invalidated Obviously the actual integration of Java fragments into the communications part of ϖ, and the imposition of restrictions on these Java fragments, is rather complex and as such is covered in the implementation chapter

Finally, the complete separation of computations from communications results in implementations of systems that are very readable and understandable, which is ideal for comparing ϖ implementations to π-calculus specifications

As previously mentioned the integration of the two disjoint aspects of ϖ is achieved, in part, by the dual nature of data items in ϖ They can be either names or objects depending on the context in which they are viewed in This ability to pull objects up from sequential computations and transform them into names for use in communications, and likewise the ability to push names into computations and transform them into usable objects, is what makes sequential computations possible, and powerful in ϖ However this duality of data items in ϖ does have the consequence of causing names to be stateful - a concept not present in the π-calculus

4 2 2 Mobility and Channels

The concept of mobility is a common one in process calculi Mobility of one form or another exists in most process calculi and the introduction of the concept of mobility was an attempt to capture the dynamic nature of distributed concurrent systems

Mobility comes in many guises, the two main forms being mobile agents and mobile links When the notion of mobility was being mulled over some choose to perceive mobility as series of agents that were free to migrate from machine to machine while maintain the same links between these processes Others choose to imagine a world in which the agents of a system remained fixed in position but the links between these agents were constantly changing It is this latter form of mobility that is present in the π-calculus, and it is this form of mobility that is also present in ϖ While the argument could be made that the presence of both forms of mobility in ϖ would be beneficial, implementation issues surrounding the migration of processes put this idea beyond the reach of this implementation

Obviously since the mobility property stems from the links, or channels, in a system, the implementation of these links is of vital importance and every endeavour must be made to ensure that the operation of this links is as close as possible to their behaviour in the π-calculus First and foremost channels should be allowed to be shared amongst agents of a system, they should be able to be learnt by agents that did not previously know of them - in other words they should facilitate the π-calculus concept of mobility Secondly, it should not be necessary to know which agent is "at the other end" of the link It should be possible to send a message on a channel without knowing which agent, if any, is "listening" on the other end Finally, channels *should* be synchronous in nature The majority of implementations of the π-calculus (Nierstrasz et al n d , Wojciechowski & Sewell 1999, Pierce & Turner 2000*a*) insist on forcing channels to operate in an asynchronous nature which restricts the usefulness of channels somewhat significantly Channels in ϖ operate in a fashion identical to the behaviour of their monadic π-calculus cousins

4 2 3 Distribution of ϖ systems

Implementations of systems specified in the π-calculus are generally intended to be deployed in a distributed fashion This necessity was recognised at an early stage in the development of ϖ and as such ϖ caters for such distributed and concurrent systems by providing a mechanism for deploying systems in an arbitrary distributed manner

This mechanism is provided via the low levels of the implementation of ϖ rather than via any language construct or feature of the language Since the apparatus that provides the distribution of ϖ systems exists in the implementation of ϖ it is more fitting to postpone detailed discussion on this feature of ϖ until the actual implementation of ϖ is delved into in greater detail m a later chapter However it is worth mentioning at this stage that ϖ allows the

distribution of processes in a system Processes in a system do not know the location of the other processes in that system, in fact they do not know what processes even exist in that system It is the implementation of channels in ϖ that facilitates the distribution of processes in a ϖ system

4 3 Example System

In general a ϖ system consists of a *System*, one or more *Process*es and possibly some Java *Code* blocks, where a Java Code block is a mechanism for the simple and quick re-use of sequential computations - much like a method in Java The *System* specifies which of the *Process*es are at the top-level, i e must be started by their environment A *Process* may perform various actions, start other *Process*es and invoke Java *Code*

The following is a very basic ϖ *System*, it allows two users to communicate over shared channels While this example is very simplistic it does demonstrate various key aspects of ϖ - replication, sequential computations, communication over channels, and the integration of Java code into ϖ, while also giving a "feel" for what can be accomplished in ϖ

The *Ytalk* system consists of two top-level processes that must be started by their environment Within the scope of these two processes are two channels that the processes will use to communicate on Each of these two top-level processes start two more processes, but this time in a replicated fashion, one process for reading messages, one process for sending messages These two common processes both perform the necessary channel actions and Java actions to allow the two users to communicate with each other

4 3 1 Abstract syntax

$$(vab)((!(f_{readMsg}()(msg)\ \overline{b} < msg >) \mid !(a(msg)\ f_{printMsg}(msg)()))$$
$$\mid(!(b(msg)\ f_{printMsg}(msg)()) \mid !(f_{readMsg}()(msg)\ \overline{a} < msg >)))$$

4 3 2 Concrete syntax - Code

```
System Ytalk
{
        Channel a,b

        (ProcessA(a,b)|ProcessA(b,a))
}

Process ProcessA(in,out)
{
        /*Start a sub-process to handle incoming messages
```

```
         *and another one for outgoing messages*/

         ('SendMessage(out)|'ReadMessage(in))
}
Process SendMessage(out)
{
        /*Input message via Java-code and then send it*/
        <readMessage>()(msg)
        out<msg>
        0
}

Process ReadMessage(in)
{
        /*Input message via channel and then
         *print it via Java-code*/
        in(msg)
        <printMessage>(msg)()
        0
}

Code readMessage()(message)
{
   /&
   try
   {
     InputStreamReader isr = new InputStreamReader( System in ),
     LineNumberReader lnr = new LineNumberReader( isr ),
     message = new String(lnr readLine()),
    }
   catch(Exception e)
   {
     e printStackTrace(),
   }
   &/
}

Code printMessage(message)()
{
        /&
        if(message getClass() getName() equals(''java lang String''))
        {
                System out print(''Other  ''),
                System out println((String)message),
        }
        &/
}
```

Explanation

The YTalk system consists of two top-level process instances, both of which are instances of *ProcessA* that must be started by their environment, i e by the users that wish to use them Within the scope of the system are two channels that will be used by the two halves of the system to communicate on

As can been seen in the above example possibilities for errors to occur in sequential computations exist If an error should occur this results in the process that is performing the sequential computation to block, i e it ceases to execute

The bodies of these two top-level processes are identical, they both start another two processes *SendMessage* and *ReadMessage* in a replicated fashion, that is an arbitrary number of instances of these processes are started depending on demand

The purpose of the *SendMessage* process is to use a Java-code fragment to obtain a message from the standard input and to "pull up" this object from the Java-code into the communications code and to then transmit this new name on one of the shared channels This shared channel will link this instance of *SendMessage* to an instance of *ReadMessage* in the other half of the system

The *ReadMessage* process reads a name in over a channel, which links to an instance of *SendMessage* in the other half of the system This name is then "pushed into" a sequential computation that transforms the name back into the original message and outputs it to the standard output

During the execution of the system the actual work will be achieved by the interactions between various instances of the replicated processes, *SendMessage* and *ReadMessage*, over the channels that are shared between the two halves of the system, the *ProcessA* half and the *ProcessB* half

4 4 Language design decisions

Implementing a programming language requires that a series of decisions and compromises be made on the way from the initial conception of the desired properties of the language to the final result yielded at the end of the process The design and implementation of ϖ was no different in this respect Decisions and compromises were necessary at both the language design and the language implementation phases

4 4 1 Sequential Computations

One of the first decisions that had to be made with regard to the language design of ϖ was related to the mechanism that was to be provided by ϖ for

performing sequential computations Two approaches to providing this mechanism were considered The first approach that was considered was the creation of a notation that would encompass all aspects of performing sequential computations in ϖ This approach would have resulted in all aspects of ϖ systems falling under one set of syntactic and semantics rules, as well as imposing tighter controls on the actions possible to be performed in sequential computations

However the implementation cost of this approach meant that an alternative approach to providing the mechanism for performing sequential computations was required This alternative approach involved the re-use of an existing programming language for performing the sequential computations in ϖ While this alternative approach may not have been the originally desired approach, it is felt that it still allows the primary goals of ϖ to be achieved

A complication that resulted from this decision was reconciling the strongly typed Java programming language and the weakly typed communications aspect of ϖ The only approach to solving this problem that could be found what to equate names to the superset of objects that are available in Java, and to equate channels to a specific type of object This approach facilitates the reconciliation of the two conflicting typing systems and allows the integration of computational code in the communications code with only the minimum of impact to the desired syntax and semantics of ϖ

4 4 2 Names and channels

In ϖ not every name can act as a channel, names must be explicitly declared as channels if they are required to act as channels However in the π-calculus each and every name may act as a channel This disconnect between ϖ and the π-calculus is perhaps one of the most significant compromises that was required to be made in the design of the ϖ language It was originally desired that all names in ϖ would have the capability to act as channels but a direct consequence of the decision made with respect to sequential computations was that a mechanism for allowing this could not be devised

Chapter 5

ϖ - The implementation

The syntax and semantics of ϖ, as previously described, outline the appearance of ϖ processes and systems, and the interaction that may occur between a series of these processes when they are constructed as a ϖ system While the formulation of these syntactic and semantic rules is a significant milestone in the development process of the ϖ programming language, it forms merely one half of the entire set of deliverables necessary for the creation of the ϖ programming language The second half of the development process revolves around the actual implementation of the language, which is the transformation of the definition of the implementation and its execution provided by the semantics into a concrete and complete programming language

The implementation of ϖ must take into account a number of requirements in addition to those implicit to any programming language with the previously described syntax and semantics These additional requirements contribute significantly to the complexity of the final implementation and this complexity is reflected in the size of the implementation and the number of technologies required to create it

The ϖ implementation must supply a mechanism to transform valid ϖ code into an executable form, and it must provide an environment in which the execution of this code can take place To further complicated matters one of the demands made of ϖ is that it should allow the creation of modularised systems, systems which can be distributed and concurrently executed Further still, the functionality should be provided which allows real-time communications to occur between these various components of these distributed and concurrently executing systems Given the desire for the channel based communications to be synchronous in nature real-time communications are a requirement

This implementation, which should satisfy the above requirements, consists of two main parts - the compiler and the runtime libraries Both these aspects

are required for the generation of systems that are structured in the required fashion and behave in the desired manner The compiler generates code that creates such systems, and this generated code relies heavily on the libraries to provide the necessary functionality, as well as aspects of the runtime environment

As the ϖ implementation is large and rather complex, and not every part of it is directly related to providing the desired functionality of the language As such a prudent approach to the examination and discussion of the implementation of the ϖ programming language is the description of each aspect of the desired functionality followed by an explanation of how these aspects were provided, rather than an investigation into the operation of the programming language in its entirety

5.1 Required Functionality

The topic of investigation in this section is not the structure of ϖ processes and systems, nor the behaviour of these entities, but rather the underlying infrastructure that facilitates the creation and operation of these processes and systems - the portion of ϖ that is "under the hood" so to speak This infrastructure can be divided in a few main categories - distribution, processes, channels, computations and the environment

5 1 1 Distribution

One of the primary desired properties of a ϖ system is that it should be capable of being executed in a distributed fashion Components of a system should be able to be deployed in an arbitrary topographical arrangement and it should be possible to make the decision as to this arrangement at run-time rather than at compile time In order to cater for these requirements a number of sub goals are required to be satisfied

For a ϖ system to be capable to be distributed over a series of machines it must first be possible to identify and separate the various parts of the system that could be distributed As such it is required the executable modules yielded by the compilation of a ϖ system must be independently executable The *only* dependency that one module, or node in the system, should have on another node is to facilitate the completion of the synchronous communications between processes

If a ϖ system, when operating in a distributed fashion, consisted of merely a number of standalone applications, each executing in complete and utter isolation, then the act of distributing a system would be a pointless one As touched

upon above, each "site" in a distributed ϖ system must to able to interact with other sites, that is the capability for inter-site communications must be present The importance of communications between sites in a ϖ system cannot be overstated as it forms one of the lynch-pins of the execution of ϖ systems It is worth mentioning that each "site" in a distributed executing a ϖ system is simply a ϖ process that resides at the top of the systems process hierarchy

5 1 2 Processes

ϖ systems are constructed using processes as the basic unit of construction A logical extension of this is that the basic unit of execution for ϖ systems should be the process The execution of ϖ systems is completely process oriented and every single item that can be executed is a process Therefore the ϖ implementation must provide a means to transform the source for a process into an executable object While the behaviours exhibited by processes are merely consequences of the semantics that define them, the issues surrounding incorporating these semantic rules into the ϖ implementation is a non-trivial task and deserves further mention

In addition, one of the properties that makes the ϖ syntax and semantics so powerful and expressive is the ability to concurrently execute processes, i e allowing processes to run in parallel Obviously the underlying implementation of ϖ also has to support this notion of concurrently executing processes, whether this execution is occurring on one machine, or is distributed over a series of machines

Replication is a massively useful tool in the theory behind the π-calculus and the ϖ programming language The ability to have an infinite number of identical copies of a process, and to have each required instance running just as you need it, allows the expression of processes that are otherwise complex, lengthy and error-prone, in a few lines of simple, self-explanatory code However serious issues surrounding the implementation of this form of replication are immediately obvious Overcoming these obstacles while still maintaining the concept of replication is most certainly a non-trivial task

The ability to have concurrent execution of processes, the provision of a mechanism to replicate these processes, the possibility of distributing these processes, and the capability for inter site communications are not the only requirements of the ϖ infrastructure Processes also have to be able to interact with each other without knowing of each other, i e there should be knowledge-less inter-process communications, which is that a process should not be concerned with what process is on the other end of a link, but rather it should be satisfied that there is another end to the link This requirement, in conjunction with the

necessity for links in ϖ systems to grow and die in a seemingly organic manner demands a complete and reliable implementation of these links - channels

5 1 3 Channels

Channels are the workhorse of the ϖ implementation They provide mechanisms that supply a significant proportion of the functionality required of the ϖ implementation Virtually every aspect of the ϖ implementation that is not directly concerned with the execution of processes and the performance of computations is provided, either partially or completely, by the implementation of channels

Chief amongst the solutions provided by channels is the solution to the problem of distribution As previously mentioned in this section, support for distribution is a key requirement of the ϖ implementation and ϖ channels provide practically all of the functionality required to support this distribution of systems As a mechanism for the provision of distribution in ϖ systems the implementation of channels must facilitate communications between the various processes that will comprise a ϖ system

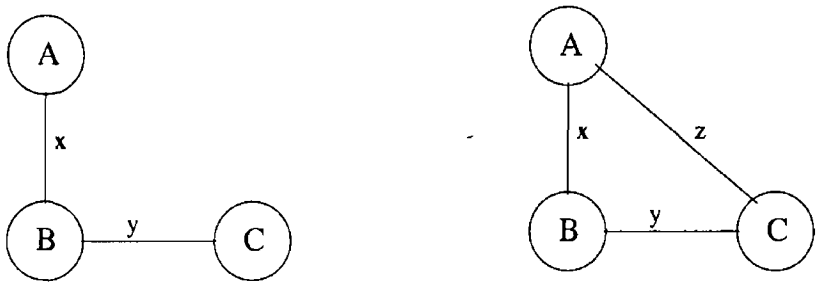

Figure 5 1 Growth of a system

One of the key purposes of channel based communications is to enable the growth of links in a system between processes of that system, for example Fig 5 1 ϖ channels, like their π-calculus cousins, must be capable of both transmitting other channels and also of being transmitted themselves By possessing both these properties ϖ channels can make the seemingly organic expansion, and reduction, of connections in ϖ systems possible

All communications between processes in a ϖ system must occur via ϖ channels Consequently the implementation of these channels must be robust and reliable Data cannot be lost, communications cannot be left half completed, and the behaviour of these channels must be consistent The fact that ϖ channels are synchronous in nature, as well as the availability of the choice operator in ϖ, means that the possibility of partially completed communications is a very

real obstacle, one which must be overcome if the implementation of channels is to be usable in any fashion Another logical conclusion, given the importance of channels to the ϖ implementation, is that ϖ channels must be stable and robust Channels must be capable of coping with high levels of usage and significant loads and they must also remain operational even under the most extreme of conditions

Another noteworthy aspect of channels is that at any one time during the lifetime of a channel multiple read requests may be made of a channel, while simultaneously multiple write requests may also be being made Synchronous channels can, by definition, only accommodate one read and one write request at a time It is therefore a requirement of the implementation of channels that it can accommodate multiple read and write requests occurring simultaneously and that it can process these requests in a non-deterministic and guaranteed fashion, Fig 5 2 However, further constraints are placed upon the operation of channels in that the operation of these channels must always be deadlock free When a channel is used in conjunction with a summation in ϖ a read operation can effectively be "backed out of" That is a process can indicate its readiness to recieve information on a specific channel and then revoke that indication should another channel complete a read operation first Deadlocks could occur if the implementation of channels did not restrict the conditions under which processes are "backed out of" as a process could, potentially, back out of all read operations and be left idle with no possibility of resuming execution, i e deadlocked

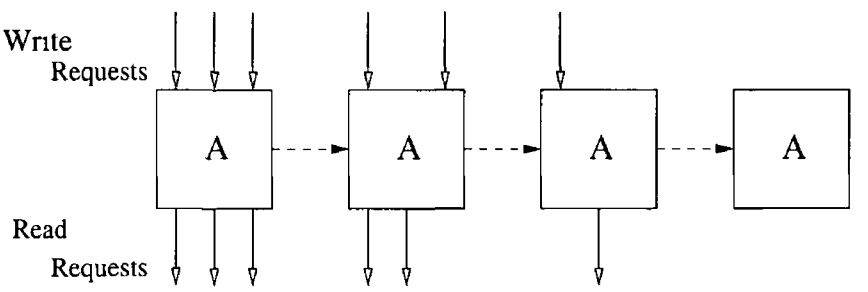

Figure 5 2 Processing of requests

Additionally the operation of these channels must be as transparent as possible, and that their operation must appear intuitive to someone familiar with traditional π-calculus channels While these requirements may seem quite trivial, they are still necessary to allow the complete ϖ implementation to remain comparable to the π-calculus

5 1 4 Computations

Additionally, what use are mechanisms for allowing concurrent, distributed and replicated processes, and methods that facilitate the communications between these processes, without a way in which to carry out calculations - to have something *to* communicate Some purpose must be given to these communicating distributed processes The final task is to allow computations to occur, and to facilitate the communication of the results of these computations between the various processes

5 1 5 Environment

ϖ systems execute in the Java execution environment However this execution environment "as is" is not sufficient to meet the requirements of executing ϖ systems Additional demands such as the initial setup and synchronisation and also the termination and clean-up of ϖ systems are made of the execution environment These demands must be met by providing a ϖ execution environment which sits on top of the Java environment This new execution environment is also responsible for enforcing the ϖ communications model

5 1 6 Summary

Taking the required functionality outlined above into account, the examination of the ϖ implementation will focus of the following topics

- Channels and the distribution of processes
- Channels and the synchronisation between processes
- Channels and the communications between processes
- Channels and the summations in processes
- Processes and the execution of these processes in a concurrent fashion
- The replication of processes
- The invocation of processes
- Performing sequential computations via in-lining methods and code blocks
- The execution environment and the initial setup and synchronisation of processes and channels
- Termination of systems in the execution environment

- Security, enforcement of ϖ communication restrictions in the execution environment, i e channels are the only available mechanism for communication between processes

5.2 Provision of Required Functionality

5 2 1 Channels

When considering how best to implement distribution in ϖ it is worth reflecting on what exactly will be distributed and how the distributed entities will interact In a ϖ system the distributable entities are the top-level processes of that system, where a top-level process is one which resides at the root of the process hierarchy, one which is invoked by a user rather than another process These top-level processes, and indeed all processes, can interact with other processes in two possible ways Firstly, a process may invoke other processes The invocation of another process results in that process executing on the same site as the "parent" process and as such this form of interaction is not concerned with the distribution of systems as only the invocation to top-level processes can affect the topology of a system It was originally desired that all processes in a ϖ system could be distributed in an arbitrary manner regardless of their position in the processes hierarchy for a system The management of this fine grained process distribution would have to be either manually managed via configuration files or dynamically managed by a distributed load balancing mechanism On the grounds that the first approach would be too cumbersome and awkward and the second approach too complex and beyond the scope of this work the more restrictive, and realistic, approach of only allowing top-level processes to determine the topology of the system was the approach taken However the second form of interaction, communications over channels, is very much concerned with distributed interaction as the processes communicating over these channels may be residing on separate host machines, Fig 5 3

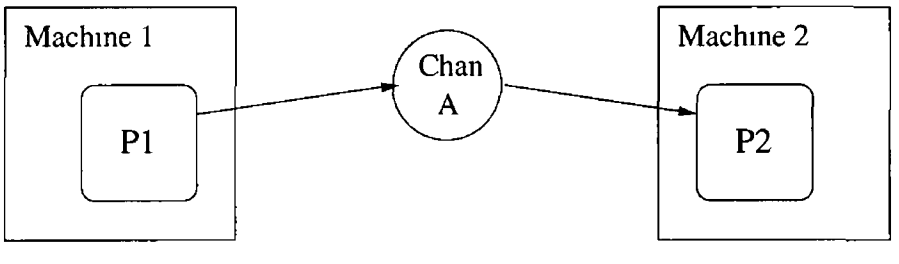

Figure 5 3 Distributed interaction

Seeing as channels are an intricate part of every communication between

possibly distributed processes it seems only fitting that the onus should be placed on channels to facilitate this distribution In order to make these distributed communications possible a channel must be visible to all processes that are required to use it, and the physical location of the actual channel, (for it must reside somewhere), should have no impact on how the various distributed processes interact with it

Java RMI

The interaction between applications running in a distributed fashion has become so commonplace that Sun Micro-Systems extended the Java programming language to include a technology called Java Remote Method Invocation, or Java RMI Java RMI is heavily used in the facilitation of communications over channels in ϖ and as such a brief overview of Java RMI is required[1]

In Java RMI remote objects are created by servers and the server makes references to these remote objects available These references may be passed around the distributed application and clients can use these references to invoke methods on the remote objects as if they were local objects For a client to use a reference to a remote object it must first obtain the reference by one of two methods It can get a copy of the reference by either looking the object up in Java RMI's simple naming service known as *rmiregistry*, or by receiving the reference as an argument or as a return value Once the remote object's reference has been obtained it can be passed around applications just like any other object, and more importantly this reference behaves as if it was the actual remote object itself Java RMI provides the mechanisms necessary for the server and clients to communicate and consequently allow the reference to behave as the remote object

Channels and Java RMI

Java RMI provides a mechanism for remote objects to appear local via references, and also provides two ways to discover references to these remote objects This is exactly what is required to implement ϖ channels The use of multiple immutable references which all refer to the same remote object allows the ϖ model of distribution to be implemented in a transparent and intuitive manner The ability to obtain references either by lookup or by parameter passing also permits the fundamental differences between top-level processes and all other processes to be overcome, that is that top-level processes are started by the user and not another process and as such cannot obtain references to remote

[1]More detailed information on Java RMI can be obtained on the website http //java sun com/products/jdk/rmi/

objects by parameter passing All in all Java RMI is a vital tool required for the implementation of ϖ channels

By using Java RMI in the implementation of ϖ channels and by making the Java class that represents ϖ channels implement the Remote object interface the task of allowing distributed processes to interact is greatly simplified Processes now use local references to remote objects, which represent channels, to communicate with each other Therefore from the perspective of a process there are no remote interaction occurring, merely the invocation of methods on local objects

Example

Process A creates a channel C Process A then obtains a reference to this newly created remote object which it then sends over an existing channel to Process B Both Process A and Process B can now use their corresponding references to the remote object, which represents channel C, to interact with each other, Fig 5 4

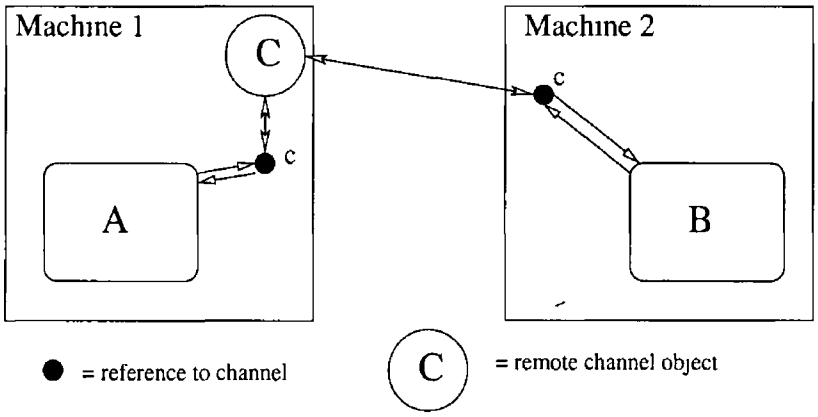

Figure 5 4 Distributed interaction

In the above example the remote channel object 'C' is shown to reside on a specific machine, Machine 1 This is because the process in which the channel associated with this remote channel object was created also resided on Machine 1 The remote channel object will reside on this machine until termination of the system or Java RMI's garbage collection removes it

One of the advantages of ϖ channels being accessible as references, and as if they were local objects, is the ease in which inter-process communications can be implemented When a process wishes to write some data to a ϖ channel it simply invokes the *write* method of the local reference and supplies it with the relevant data, Fig 5 5 The underlying Java RMI mechanisms handle the transmission of the data to the actual remote object

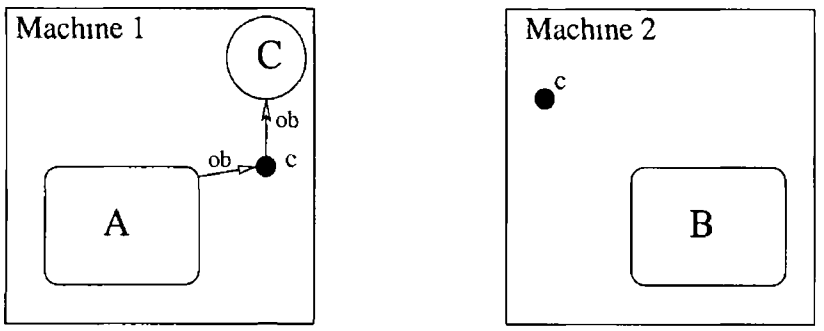

Figure 5 5 A Write Request

Likewise when a process wishes to read data from a channel it calls the *read* method of the local reference which will return data when it is available, Fig 5 6 Again Java RMI deals with the actual transmission of data from the remote object

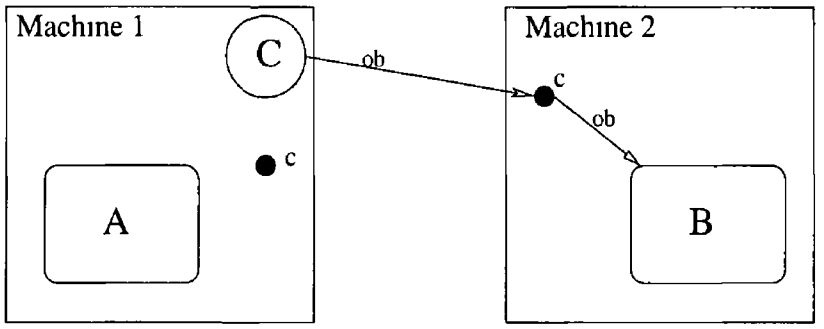

Figure 5 6 A Read Request

The previous description of inter-process communications over channels was a simplification It proved a useful example to outline the rough concepts behind ϖ channels However further issues surrounding the communications between processes over channels exist Amongst these issues is the implementation of the synchronous nature of ϖ channels Given this synchronous nature it is an obvious necessity that when a read request is made of a channel that there must be a corresponding write request, i e something must have put the data on the channel m the first place, and if no data is present then the read request is forced to wait for some to become available In an asynchronous implementation of channels this would be the only requirement made of channels with regard to their behaviour, that is in order for a read operation to complete there must be data present There would be no restrictions placed on write operations, they would merely write their data to the channel regardless of whether there is a corresponding read operation ready to complete the transaction or not and

continue on. However ϖ channels *are* synchronous in nature and as such it is essential for a write operation to occur that there must be a corresponding read operation ready to occur as well. In short no channel operation can occur without the opposite operation also occurring on the same channel, Fig 5.7. This problem is solved using a series of locks and notifies on the Java objects used to implement channels. Further detail of this solution is provided later in this chapter.

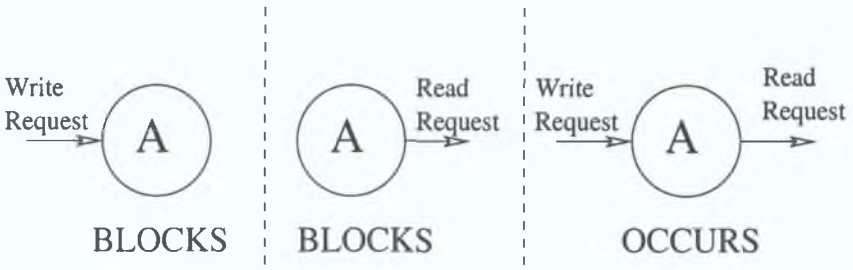

Figure 5.7: Synchronous operation of channels

However the behaviour of channels is further complicated as a result of the possibility that multiple read and write requests may be made of a channel at the same time, Fig 5.8. The implementation of channels must allow multiple requests of both kinds to be made simultaneously and to process these requests in pairs, one read and one write, and also maintain a queue of requests that remain to be processed.

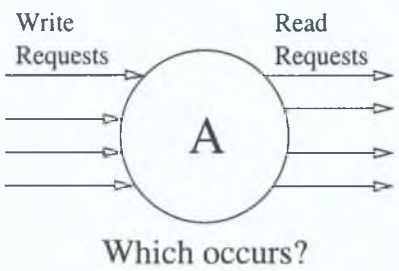

Figure 5.8: Multiple Requests

The first requirement in implementing the read and write methods of ϖ channel is that only one request of either type may be active at any one time. That us a read operation cannot be occurring at the same time that a write operation is occurring and vice versa. Likewise only one read request may be occurring at any one time, and only one write operation can occur at the same time. Conveniently the Java programming language provides a mechanism to ensure that this happens, the *synchronize* statement. The contents of both

summation, $a(x)P + b(z)Q$, means that the invocation of processes depends on the occurrence of associated actions, $a(x)$ occurring starts P and $b(z)$ occurring starts Q The problem arises here from the fact that the above read algorithm does not provide a mechanism for a read operation to "back out" without completing and indicating that the operation has successfully completed This mechanism is necessary in order to avoid undefined and unexpected behavioural consequences Out of all the input actions in a guarded summation exactly one of these actions should be allowed to complete and the remainder of the input actions should be able to "back out" without having any negative effects on the behaviour of the system This new mechanism is catered for by the provision of a *conditionalRead* method in channels which should be used in summations

Conditional Read algorithm

```
Synchronize readLock
        Synchronize actionLock
                if data present
                        set task done
                else
                        wait on actionLock
                        set task done

                if ! stopped
                        wait until stopped

                if task done by me
                        notify on actionLock
```

Conditional reads from ϖ channels require the use of two additional classes - the *Task* class and the *Reader* class Both of these classes play pivotal roles in allowing conditional reads to occur The use of the *Reader* class is necessary to orchestrate the multiple blocking requests to be made of numerous channels which are required to allow conditional reads to occur Some way of multi-threading these requests is necessary and the *Reader* class provides this functionality

The reasoning behind the necessity and functionality of the *Task* classes is rather more complex In brief the *Task* class is responsible for determining which input action is the one that will occur, it is also responsible for informing that input action that it should occur and finally the *Task* class is responsible for terminating all reader threads successful or otherwise

5 2 2 Processes

Given that in ϖ the implementation of channels is responsible for providing the functionality required for communications and synchronisation between the distributed processes of systems the only topics that are related to processes that

methods are enclosed in *synchronize* statements and this ensures that the desired behaviour is provided In addition the *synchronize* statement also provides the automatic queuing of other requests

The second priority in implementing these methods is to ensure that on completion of a channel operation that both read and write methods complete at the same time regardless of which request was made first There are two possible orderings of the requests and each one must be catered for

Write first, read second

The write method sets the data and then waits for the data to be read The read method gets the data, indicates that the data has been read and both methods complete

Read first, write second

The read method attempts to get the data, none is present so it must wait for some to be made available The write method now sets the data, it indicates that the data has been set which results in the read method waking and reading the data Finally the read method indicates that the data has been read and both methods complete

In both these scenarios the following algorithms result in the desired behaviour, these algorithms were produced as the result of much analysis of the problem at hand and many prototypical implementations, in hindsight time and effort could have been saved by timely consultation of literature related to concurrent programming

Write algorithm

```
Synchronize writeLock
        Synchronize actionLock
                set data
                notify on actionLock
                wait on actionLock
```

Read algorithm

```
Synchronize readLock
        Synchronize actionLock
                if data present
                        get data
                else
                        wait on actionLock
                        get data
                notify on actionLock
```

Using only the above algorithms as the basis for an implementation of the read and write methods of ϖ channels would suffice were it not for the presence of guarded summations in the syntax and semantics of ϖ If we recall a guarded

remain to be discussed are how processes execute concurrently, how processes are replicated and an explanation of the different ways in which ϖ processes can be invoked

Concurrent Execution

Concurrent execution in ϖ involves the execution of an arbitrary number of process in parallel Given that the primary intended use of ϖ is in the implementation of distributed systems, it is extremely likely that each site in a ϖ system will play host to a number of processes, all of which are required to be running in parallel to each other, some method of multi-threading the execution of these processes is necessary Once again the Java programming language provides a mechanism which aides us in overcoming yet another problem The Java programming language provides a way to create multiple threads, where a thread is a single distinct strand of execution, and to have these threads executing concurrently By making each process in ϖ a Java thread and by starting these threads in a concurrent manner it is possible for the concurrent execution of processes to occur in ϖ However given the nature of Java multithreading this would not be considered "true" concurrency from a π-calculus perspective, however to the user it would appear so

Traditionally a problem existed with having multiple threads executing currently in Java These threads lacked a guaranteed and reliable method to communicate and synchronise with other threads However the use of ϖ channels in the Java multi-threaded environment has solved both the problems of communications and synchronisation between concurrently executing threads

Replication

Replication in the strictest π-calculus interpretation is not feasible from an implementation point of view The idea of an arbitrary, possibly infinite, number of instances of a specific process all ready to run, in fact all running and merely waiting to interact with other processes, Fig 5 9, is not a concept that is reconcilable with real world computing and computers As a result it was necessary to implement replication differently This different approach to implementing replication still results in the same casual observable behavioural properties but a more realistic approach was necessary to achieve these properties Instead of having an arbitrary number of processes ready for execution ϖ replication only ever has exactly one more instance that what is presently needed executing This approach allows the replication process to behave in the same manner but it is not as resource intensive

This approach is made possible as a result of the manner in which the Java

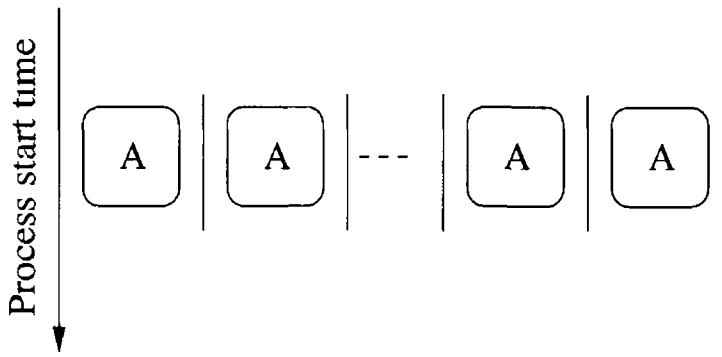

Figure 5 9 Pi-calculus replication

threads that represent ϖ processes are generated Since each thread is tailored specifically to each individual process it is possible to add the capability for replication to each process by ensuring that each instance of a replicated process invokes exactly one other instance of itself after it performs its first action, be it an input, an output or a Java action, Fig 5 10 This results in there always being one more instance of a process running than is currently required By always having one more than necessary future demand for interaction with additional instances of a process is always catered for However this approach does result in issues surrounding the termination of ϖ systems, these issues will be investigated later

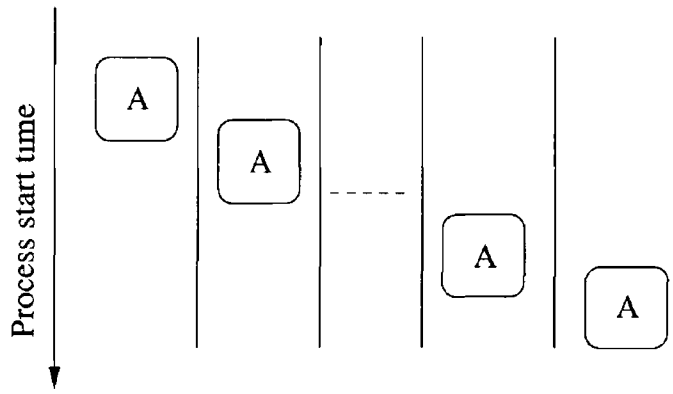

Figure 5 10 ϖ replication

Invocation

There are a number of different scenarios in which a process may invoke, or start, another process A process may be started as the only "child" process of

another process, or a process may be started in a replicated fashion, or a process may be started (and possibly replicated) in parallel to a series of other processes These scenarios for starting processes have been previously explained, but the workings of two additional scenarios in which processes may be started have not yet been touched upon - starting processes via a choice statement and starting processes via a match statement

Match start

A match statement consists of a number of condition statements, each with an associated process invocation statement and also a default process invocation statement The conditions are evaluated from left to right and the first condition statement that is satisfied has its associated process invocation statement performed Should none of the condition statements be satisfied then the default process invocation is performed

When a condition statement is being evaluated the actual testing of equality of names is done by reference not by value As such condition statements, and indeed match statements, are mainly of use when comparing channel names as opposed to non-channel names

Choice start

The choice statement is a very useful and powerful statement It allows the execution of a system to be affected by the occurrence, or non-occurrence, of various input actions A choice statement consists of a number of input actions, and each input action has an associated process invocation statement Once one of these input actions occurs, and only one of them can ever occur, the process invocation statement associated with the input action is performed This triggers a whole new set of process instances to be started The possibly complex task of implementing choice statements was greatly simplified as a result of the manner in which ϖ channels were implemented By using the conditional read functionality of ϖ channels and the existing functionality for invoking processes the implementation of input guarded summations in ϖ was achieved

5 2 3 Computations

The ability to integrate strongly typed computations into an untyped communications framework is one of the main attractions of the ϖ implementation While these computations can be performed in one of two ways, inline or code-blocks, the majority of the issues surrounding computations are common to both

The major issue that arises from this integration is the typing problem, the communications code is untyped - where everything is a name, and the computational code, written in Java, which can contain any combination of types from a very rich set of types This problem is overcome by using more features of the Java problem language - class casts and the *Serializable* interface

When a name is passed in a computation in ϖ the Java code contained in the computation can access the name as a *Serializable* object Also, regardless of what the computation does, all names created in the communications code are also *Serializable* objects As the only other way to create a name is to create a channel, and all channels are also *Serializable* This ensures that all objects in the communications code are only ever of one type - *Serializable* This allows the strongly typed aspect of computations to be reconciled with the untyped communications

This of course requires the Java code inside a computation to cast its parameters into more varied types Achieving any task of worth in a Java program that works solely with Serializable objects would be rather difficult On first glance this may seem like a serious problem as the possibility for class cast exceptions exists However on closer inspection the risk is no greater than extracting and using the various elements of a heterogeneous Java Vector - care must simply be taken in writing and testing systems and processes

Names, distribution and consistency

When a name is created, either by a sequential computation or by creating a new channel, and communicated amongst various distributed processes the task of ensuring the consistency of this name across these sites becomes a formidable one A far simpler and neater solution to the problem of ensuring consistency of names across remote sites is to insist that all names are immutable By doing so the functionality to reflect changes in names made in one site on all other sites is not required Now once a name is created it cannot be changed, it can be "forgotten" and replaced but never changed

In order to enforce this policy of immutable names it is necessary to take a snapshot of all names that a computation can access, for it is only in a computation that the possibility of altering names arises, before the computation is performed, and restoring this snapshot after the computation has been completed, Fig 5 11 An additional beneficial consequence of this approach is that no unexpected side effects can arise from the computation of calculations

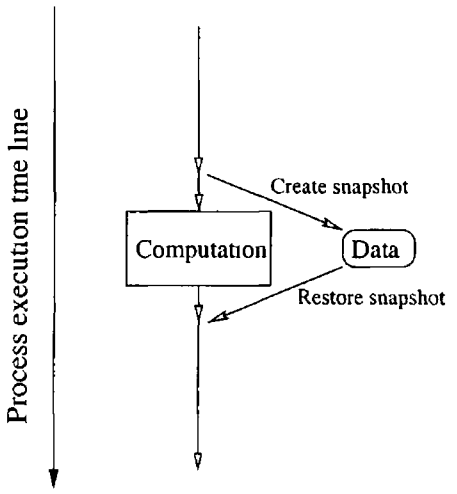

Figure 5 11 Storing and restoring names

Failure and fault tolerance

A process in a ϖ system can fail for a number of reasons, these include network issues, I/O problems, and exceptions and errors thrown from Java fragments The result of a process failing is the same regardless of the reason for failure - the process blocks More specifically the process in question terminates and frees any resources that it may be using However the process is no longer in a state to interact further with other processes in the system It is possible that the lack of these further interactions will have no effect on the other processes in the system, however it is more likely that the non-occurrence of channel output/input as a result of the termination of the failed process will cause other processes in the system to block, i e to wait indefinitely on specific channel actions These blocking processes will cause the system to halt No mechanism for the notification of such failures nor the recovery from such failures exists

5 2 4 The Environment

In order to allow the execution of ϖ systems it is necessary to provide an execution environment The ϖ execution environment is responsible for the initial pre-execution setup of systems and for the post execution termination of the various sites as well as enforcing the communications model These setup and termination phases rely on the use of a lightweight centralised application, the location of which is known to all top-level processes

Setup

Before a ϖ system can begin execution some setup is required This setup is mainly concerned with the channels that are shared amongst the top-level processes Since these top-level processes cannot receive references to the remote objects that represent channels by parameter passing they must obtain these references by the look-up method instead In order to look-up a reference it is necessary to know the site on which the remote object resides Once this information is known the actual act of look-up is rather simple The initial setup phase of the execution of a ϖ system deals with the distribution of this information to the various concerned top-level processes However, before the information about the actual physical location of channels can be dealt out the responsibility for these channels must first be allocated This allocation is done using a specific allocation algorithm which takes into account a number of factors before allocating responsibility for a channel These factors include whether the process uses the channel, whether the process is replicated, and the existing load on the site that hosts the process Preference is given to non-replicated processes with minimal loading of their sites that use the channel in question Once all the setup information has been distributed the actual execution of the system can begin

Termination

Termination in ϖ is not so much concerned with the termination of individual processes but rather with the complete termination of all execution on a specific site Since a site may be responsible for channels that other processes are using even though there are no more active processes on the site, it is necessary for all sites to remain "up" until all the sites in a system are all inactive When this happens all the sites that make up a system can "come down" There is, however, one exception That is a site that plays host to a top-level process that is replicated and not responsible for any channels that are shared at the highest level may terminate once all processes executing on that site finish

The migration of channels from site to site was briefly considered but was dismissed as the cost of implementing this would far outweigh the value of it

Security

In the π-calculus the only method for communicating between processes is by the use of channels It is therefore a necessity that in ϖ that the only way that processes can communicate is also by the use of channels This communication model needs to be enforced at a low level and is done so by implementing a Java security manager that monitors and regulates all network connections and

communicates and ensures that nothing is done to breach the desired communications model The Security manager class has available to it the signatures of all methods involved in any attempt to perform network I/O and by examining the collection of signatures involved in any attempt to perform network I/O the security manager class can prevent undesired network I/O

5.3 Language Implementation Decisions

As would be expected once the core language design decisions were made and the design of the language implementation started a number of decisions regarding the language implementation were required to be made While most of the design decisions resulted in the features in question being incorporated into the implementation some compromises were required to be made around a number of issues

5 3 1 Channel migration and termination

As has been previously explained when a channel is created the Java RMI remote object that represents the channel is hosted on the machine on which the creating process resides A feature that was originally desired for the ϖ implementation was the ability for channels to effectively migrate from one host to another This would allow a simpler and more robust to the termination of a site in a ϖ system As it stands all sites in a ϖ system must signal their desire to terminate before a single site can do so This is to ensure that channels that in use by processes on different sites are not affected by the termination of a specific site If channels were able to migrate from one site this problem would be avoided However due to implementation difficulties surrounding this feature it had to be descoped from the project

5 3 2 SyncServer

Presently there is a requirement for a central mini-server in each ϖ system to facilitate synchronisation at system initialisation and termination Currently a "syncserver" is required to aid in the communication of information related to top-level channels amongst the various top-level process during initialisation and in establishing agreement as to when a system can terminate completely While allowing the migration of channels as previously described would remove the requirement for the SyncServer in system termination, in the present design of the ϖ implementation there would still be a need for it in the initialisation of ϖ systems While no mechanism which would avoid the requirement for a SyncServer during system initialisation was identified it would be desired if the

need for this central server could be avoided, however it is distinctly possible that this may not be feasible

5 3 3 Channels and Security

During the initial design phase of the ϖ language implementation the idea of attempting to secure, using various cryptographic protocols, communications occurring over channels arose However it was realised that a secure system, one written using established security protocols, would be secure regardless of the medium used to transmit information between elements of the system and as such there was no real requirement to encrypt data transmitted on channels in ϖ

5 3 4 Process migration

In the present ϖ implementation processes execute *on* the same hosts as the processes that invoke them Some investigation into balancing the execution load of these processes amongst the various hosts that constitute a ϖ system was originally undertaken It was determined following this investigation that the functionality required to facilitate this migration of processes would require considerable effort and may in fact introduce some security related issues into ϖ systems As such the concept of process migration was removed from the design of the ϖ language implementation

5 4 ϖ and the classification criteria

As both the ϖ language and its implementation have now been presented it is now possible to examine the ϖ language against the classification criteria outlined in chapter three

5 4 1 Syntax and Semantics

It was desired that the syntax and semantics of the π-calculus and that of ϖ would be similar While they are quite similar there are a number of divergences between the two These differences, which have been previously discussed, do not however make the syntax and semantics of both to be irreconcilable

5 4 2 Mobility

The π-calculus concept of mobility allows processes in a system to dynamically learn of new links between elements of that system at run-time This mechanism is present in ϖ and central to the operation of ϖ

5 4 3 Synchronous vs asynchronous communications

Communications over channels can be performed in either a synchronous or asynchronous fashion While the implementation of asynchronous channels would have be significantly simpler than implementing synchronous channels the extra effort was deemed necessary and as such the ϖ channel implementation is synchronous in nature

5 4 4 Distribution

ϖ supports distributed systems However the manner in which these systems may be distributed is restricted As previously explained the decision as to the distribution of a system must be made with respect to top-level processes and cannot be made at a lower level While this should not negatively affect the execution of ϖ systems it does restrict how ϖ systems can be distributed

5 4 5 Sequential computations

ϖ allows even the most complex sequential computation to be performed in it via the use of fragments of the Java programming language While this achieves the goal of providing a mechanism for performing sequential computations it is not the most pleasing of solutions As described in the language design decisions it would be preferred if the same level of support for sequential computations could provided but via a new notation more fitting to ϖ

5.5 ϖ and the classification categories

Given the classification categories laid out in chapter three and given the results yielded when examining ϖ against the classification criteria also laid out in chapter three it becomes apparent that ϖ does not fit into any of the three categories previously identified As such the classification categories presented in chapter three must be extended to allow the categorisation of ϖ

Category 4

A programming language belonging to category four is syntactically and semantically similar to the π-calculus It provides a high level of support for the implementation of distributed systems and it also provides a mechanism for performing complex sequential computations Communications over channels in a category four programming language occur in a synchronous manner and also facilitate the π-calculus concept of mobility

Chapter 6

ϖ examples

When previously examining the various aspects of the ϖ language, they have been examined in isolation In order to obtain a true understanding of how these various components of ϖ can be used together and how they interact with one another it is necessary to observe larger, richer example systems [1]In the following example systems the following aspects of ϖ will be amongst those used and examined

- Replicated top level processes
- Use of sequential computations via code blocks
- Use of sequential computations via inline code statements
- Replicated invocation of standard processes
- Invocation of standard processes
- Channel operations - input and output
- Summations - process choice

[1]A guide to compiling, debugging and deploying ϖ system is supplied in the appendices

6.1 Example 1 - Certificate Authority

The goal of this ϖ system (A full listing of code for this system is in appendix B), is to provide an implementation of a system that allows X509 certificates to be requested by an arbitrary number of clients, and for these requests to be fulfilled by the issuing of certificates by a central static entity - a Certificate Authority

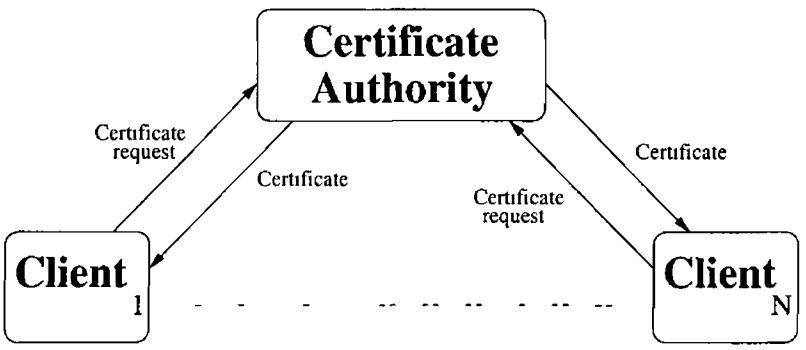

Figure 6 1 Abstract behaviour of example System 1

This type of system lends itself to demonstrating various aspects of the ϖ language, in particular the replication of top-level processes, the replication of standard processes, sequential computations and basic channel operations

The communications part of this system, the processes and their interactions, is rather simple and consists of only four processes, one of which is the System process

```
1 System Sys
2 {
3         Channel a
4
5         (CertAuth(a)|!ClientCreateCert(a))
6 }
7 Process CertAuth(cert)
8 {
9         <getInfo>()(filename,passphrase)
10        <getIssuer>(filename,passphrase)(ca)
11        (!Issuer(cert,ca))
12 }
13 Process Issuer(in, issuer)
14 {
15        in(channel)
16        channel(certRequest)
17        <issueCert>(issuer,certRequest)(cert)
18        channel<cert>
19        0
20 }
```

```
21 Process ClientCreateCert(chan)
22 {
23         <loadCACert>()(cacert)
24         <createCertificateAndRequest>()(client,req)
25
26         Channel tmp
27         chan<tmp>
28         tmp<req>
29         tmp(cert)
30         <storeClient>(cert,client,cacert)()
31         0
32 }
```

As would be expected the first step in both branches of execution are concerned with set-up and initialisation The *CertAuth* process uses two sequential computations to load and configure the data that is required so that a replicated process that will handle all the certificate requests can be invoked The first of these sequential computations, invoked from line 9, prompts a user to enter the location of the encrypted data store that contains all relevant keys and certificates required to operate the *CertAuth*, and it also prompts the user for the passphrase that will allow the data store to be decrypted and its contents used

Listing Code-block called from line 9

```
33 Code getInfo()(fn,pp)
34 {
35   /&
36   LineNumberReader lnr =
37     new LineNumberReader(new InputStreamReader(System in)),
38   System out println(''Enter the ca name''),
39   fn = lnr readLine(),
40   System out println(''Enter the passphrase''),
41   pp = lnr readLine(),
42   &/
43 }
```

Once these pieces of data have been obtained it is necessary to load and decrypt the data store (line 10) in order to create the entity required to actually issue certificates The invoked sequential computation creates a simple CA object and then uses this object to create the object which will be used to issue certificates This separation of CA and issuer is present as while there will be multiple instances of issuers, as the process that uses them is replicated, it is desired that there is only ever one actual CA This separation becomes more relevant in systems which include functionality for certificate revocation

Listing Code-block called from line 10

```
44 Code getIssuer(fn,pp)(issuer)
45 {
```

```
46  /&
47  CA theCA = new CA((String)fn,(String)pp),
48  CertIssuer ca = theCA createIssuer(),
49
50  issuer = ca,
51  &/
52 }
```

Once this sequential computation has been completed the *CertAuth* process starts a replication *Issuer* processes (line 11) It is instances of this *Issuer* process that interact with instances of the replicated client process, *ClientCreateCert*, in order to facilitate the actual requesting and issuing of certificates

The operation and interaction of these two processes occurs as follows

The *ClientCreateCert* process loads the certificate belonging to the Certificate Authority, which was distributed out-of-band, using a sequential computation (line 23) The sequential computation prompts the user for the location of the CAs certificate which it then loads as a byte[] and pushes back up into the communications The certificate is loaded as a byte[] as opposed to a Java Certificate object because of the requirement that all objects pushed into the communications code be serializable This is not checked at compile time but would rather manifest itself as a runtime error as there is no type checking of this kind, i e what can be communicated on channels, at compile time

Listing Code-block called from line 23

```
53 Code loadCACert()(cert)
54 {
55  /&
56  String filename = Client getCAFileName(),
57  byte[] cert_bytes = Client loadCACert(filename),
58  cert = cert_bytes,
59  &/
60 }
```

Once the certificate belonging to the certificate authority has been loaded the next step is for the client process to create the actual certificate request that will be sent to the CA (line 24)

Listing Code-block called from line 24

```
61 Code createCertificateAndRequest()(client,req)
62 {
63  /&
64  Client c = new Client(),
65  byte[] name = Client getName(),
66  String pwd = Client getChallenge(),
67  byte[] tmp = c generateCertificateRequest(name,pwd),
68
69  client = c,
70  req = tmp,
```

```
71  &/
72 }
```

In order to create the certificate request it is first necessary to get the fully qualified name of the intended subject of the certificate that is being requested via a method in the Client class (line 65) Following this is it necessary to obtain a challenge password A challenge password is used in the attributes of the certificate request in order to supply user credentials with the request (line 66) Once this has been obtained from the user the *Client* class is then again used to both generate the certificate request and the associated RSA key-pair (line 67) The functionality for this is supplied in a series of Java class files that were specifically written for this example Following this the newly created client object and the certificate request are pushed back into the communications code

The communications aspect of this code then creates and distributes a channel that will be used solely for this transaction (line 26) Once created and sent, this channel is then used to send the certificate request created on line 24 to the certificate authority All that remains for the client to do is to read back the issued certificate, if it was issued (line 29) and to then store the certificate along with the associated key-pair (line 30) The method that stores this information has to obtain both the location in which to store it (line 82) and the passphrase that will be used to protect the sensitive information (line 83)

Listing Code-block called from line 30

```
73 Code storeClient(cert,client,cacert)()
74 {
75   /&
76   byte[] theCACert = (byte[])cacert,
77   byte[] theCert = (byte[])cert,
78   Client c = (Client)client,
79
80   c setCertificate(theCert,theCACert),
81   String pp = Client getPassPhrase(),
82   String fn = Client getFilename(),
83   c store(pp,fn),
84   &/
85 }
```

On the other side of the transaction the *Issuer* process reads (line 15) the "session" channel that was sent by the *ClientCreateCert* on line 26 The *Issuer* process then reads the certificate request (line 16) Once read this certificate request is pushed into a sequential computation (line 17)

Listing Code-block called from line 17

```
86 Code issueCert(issuer,request)(cert)
87 {
```

```
88  /&
89  CertIssuer i = (CertIssuer)issuer,
90  byte[] req = (byte[]) request,
91
92  cert = i processCertificateRequest(req),
93  &/
94 }
```

After the certificate is created it is then sent back to the *ClientCreateCert* process

The replicated nature of the *ClientCreateCert* process and the *Issuer* process in this system allows this certificate request/issue cycle to occur as often as required

6.2 Example 2 - Certificate Authority and Service Provider

This example [2] builds upon the previous example by taking the certificate issuing infrastructure and using it to enable authentication and security in a distributed system This new system allows client processes to request the services of a service provider in a secure and mutually authenticated manner

In this example system there are three main types of entity - the Certificate Authority, the Service provider and the client The certificate authority issues certificates both for the clients and for the Service provider It also facilitates the distribution of the certificate associated with the Service provider The Service provider interacts with the Certificate authority to obtain a certificate and it then uses this certificate in a cryptographic protocol which provides the mechanism for the required mutual authentication with clients and for the secure exchange of a session key

The client aspect of the system is split into two parts One part requests and obtains certificates from the certificate authority and the other part uses this certificate in the protocol used to secure communications with the Service provider

6 2 1 The processes

In order to reuse the processes from the previous examples only a minor change was required to be made to the *CertAuth* process, and in order to accommodate the new behavioural requirements of the system two new top-level processes and some new top-level channels were required

[2] A full listing of code for this system is in appendix C

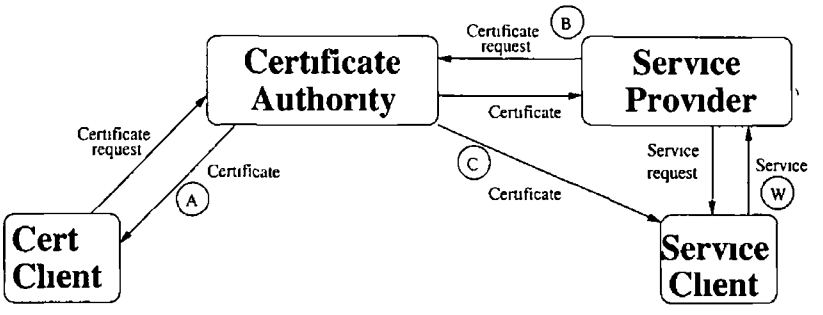

Figure 6 2 Example 2

```
95 System Sys
96 {
97   Channel a,b,c,w
98   (CertAuth(a,b,c)|'ClientCreateCert(a)|
99     ServiceProvider(b,w)|'ClientServiceRequest(c,w))
100 }
101
102 Process CertAuth(cert,spCert,spCertOut)
103 {
104   <getInfo>()(filename,passphrase)
105   <createCA>(filename,passphrase)(ca)
106   <getIssuer>(ca)(i)
107
108   ('Issuer(cert,i)|ServiceProviderIssuer(spCert,i,spCertOut))
109 }
110
111 Process ServiceProviderIssuer(in, issuer,out)
112 {
113   in(channel)
114   channel(certRequest)
115   <issueCert>(issuer,certRequest)(cert)
116   channel<cert>
117   ('DistributeCert(out,cert))
118 }
119
120 Process ClientServiceRequest(cert, work)
121 {
122   <loadCACert>()(cacert)
123   <loadSelf>(cacert)(self)
124
125   cert(spCert)
126
127   <verifyCertIssuer>(cacert,spCert)()
128
129   <createClientRequest>(spCert,self)(packet1, randA)
130
131   Channel chan
```

```
132   work<chan>
133
134   <getOwnCert>(self)(ownCert)
135   chan<ownCert>
136   chan<packet1>
137
138   chan(packet2)
139   <processServiceResponse>(packet2,spCert,self,randA)(packet3)
140   chan<packet3>
141
142   chan(encKey)
143   chan(service1)
144   chan(service2)
145
146   <extractKey>(spCert,encKey,self)(key)
147
148   <whichService>(service1,service2)(service)
149
150   Channel chan
151   service<chan>
152   chan(resp)
153
154   </&System out println((String)resp),&/>(resp)()
155   0
156 }
157
158 Process ServiceProvider(chan,work)
159 {
160   <loadCACert>()(cacert)
161
162   <createCertificateAndRequestSP>(cacert)(sp,req)
163
164   Channel tmp
165   chan<tmp>
166
167   tmp<req>
168
169   tmp(cert)
170
171   <setCertificate>(cert,sp,cacert)(newSP)
172   ('Servicer(work,newSP))
173 }
174
175 Process Servicer(work,self)
176 {
177   work(channel)
178   channel(clientCert)
179   channel(pac1)
180
181   <processClientRequest>(clientCert pac1 self)(randA)
```

```
182  <createServiceResponse>(clientCert,self,randA)(pac2,randB)
183  channel<pac2>
184
185  channel(res)
186  <processClientResponse>(clientCert,self,res,randB)(encKey,key)
187  channel<encKey>
188
189  Channel service1,service2
190  channel<service1>
191  channel<service2>
192
193  +(service1(a)(Service1(a)))+(service2(b)(Service2(b)))
194 }
```

Now instead of starting only a replicated *Issuer* process the *CertAuth* process also starts an instance of a *ServiceProviderIssuer* (line 108) This *ServiceProviderIssuer* process (line 111) is identical to the *Issuer* process that has been documented in the previous example with the exception the process invocation statement that appears at the end of it (line 117) Instead of merely receiving the certificate request from a Service Provider and sending the issued certificate back to it, this process now invokes a replicated process (line 120) whose sole function is to output the certificate that was issued to the Service provider on a channel that is known by all entities in the system This mechanism uses lazy evaluation to fill the channel with an infinite number of certificates This distribution of the Service providers certificate is required for the successful execution of the cryptographic protocol that will be used between the client that is requesting services and the service provider that is providing them

As the procedure for the requesting and issuing of certificates has previously been explained the remaining item of interest in this example is the interaction between the Service provider and the Client process that requests its services

The first task for a *ServiceProvider* to complete is the requesting and obtaining of a certificate from the certificate authority This is done in a very similar manner as the requesting of a clients certificate, lines 160 to 171 Once the certificate has been requested and obtained the actual interactions between service provider and client begin via the replicated invocation of a *Servicer* process on line 172

6 2 2 The protocol

The authentication protocol that is used in this example is a well documented (Schneier 1996) three-way protocol that makes extensive use of the X509 certificates previously issued

In the following protocol [3]

[3]In this version of the protocol the need for timestamps has been eliminated and as such

R_x indicates the random number generated by x

T_x indicates the time stamp generated by x

I_X indicates the identity of X

C_x indicates the X509 certificate belonging to x

$D_x(N)$ is the result of encrypting N with the private key belonging to x

$E_x(N)$ is the result of encrypting N with the public key belonging to x

Before this protocol can be used the certificate of the service provider must be distributed to all entities that wish to communicate with it, this is done by means of the replicated process that is invoked in the *ServiceProviderIssuer* process

Step 1

The client(CSR) generates a random number, a time stamp and some random data This random data is encrypted using the public key of the service provider(SP) which is extracted from the distributed certificate Once this is done the random number, the time stamp, the identity of the service provider, and the encrypted data are all encrypted using the private key of the client, this effectively signs the entire block of data In the system this is done in one sequential computation called from line 129 Once this is created the certificate of the client and the encrypted/signed data is sent to the service provider, line 136

$$\text{Client} \longrightarrow \text{Service provider}, C_{CSR}, D_{CSR}(M), \text{ where}$$
$$M = (T_{csr}, R_{csr}, I_{SP}, E_{SP}(d))$$

Step 2

When the service provider receives the encrypted block of data from the client along with its certificate the first thing the service provider must do is to verify the certificate was issued by the correct CA Once this test occurs the service provider then decrypts the data using the public key extracted from the certificate belonging to the client Following this the service provider must check that the value I_{SP} is in fact its own identity, and that the data $E_{SP}(d)$ can be decrypted using its own private key If these tests are successful the Service provider generates a random number and time stamp of its own and uses these to construct a message that consists of its time stamp, its random number, the identity of the client, the random number of the client and some random data encrypted with the public key of the client This message is then encrypted with the private key of the service provider and sent back to the client, line 183 This

all timestamps are 0

processing of the received data and the creation of the data packet that is to be returned to the client takes place in two sequential computations, lines 181 and 182

$$\text{Service Provider} \longrightarrow \text{Client}, D_{SP}(M'), \text{ where}$$
$$M' = (T_{SP}, R_{SP}, I_{CSR}, R_{CSR}, E_{CSR}(d))$$

Step 3

The client, on receipt of the encrypted data, line 138, decrypts the message and verifies the value of I_{CSR} and R_{CSR} and also that it can decrypt $E_{CSR}(d)$ If these tests are successful then the client encrypts the random number R_{SP} with its private key, and returns it to the service provider, line 140 The processing of the data received from the service provider and the creation of the packet that will be returned to it occurs on line 139

$$\text{Client} \longrightarrow \text{Service provider}, D_{CSR}(R_{SP})$$

Step 4

Finally the service provider decrypts the data and verifies the value of R_{SP}, line 186 After this stage mutual authentication has occurred

Following the occurrence of the authentication protocol the service provider creates a session key, also in the sequential computation invoked from line 186 In the same sequential computation the session key is encrypted with the public key of the client and signed with the private key of the service provider This encrypted and signed key is then sent to the client process along with two newly created channels

6 2 3 The service request

Once the client receives the encrypted session key, it decrypts and verifies it to yield a usable session key (line 146) The client process has also received two channels from the service provider - each channel representing a service that the service provider provides The client prompts the user as to which service it wishes to avail of, line 148 The selection of the user dictates which channel is used, that therefore which service is requested Once the service is requested, this is done by sending a channel over the relevant channel, the client process waits for a response from the service, line 152, and then prints the response to the screen For the purposes of this example the services that were requested

were kept to their most absolute simplest - i e simple strings are returned for each request

The interesting aspect of a service request takes place on the service providers part of the interaction On line 193 the service provider makes use of a summation statement - in a summation the course of execution of the system depends entirely on which input action in the statement occurs The guards in this summation are input actions involving the two channels that were sent to the client process The client process responds on one of these channels depending on which service it wishes to request As such the process that the service provider invokes depends which channel action occurs Should the client respond on the channel associated with service one then the service provider starts the process that represents service one, likewise for service two

The replicated nature of the *ServiceProvider* process and the *ClientServiceRequest* process in this system allows this service requesting/granting cycle to occur as often as required

6 3 Developing ϖ systems

In the above examples the following steps to developing the systems were followed

1 A π-calculus specification for the system was written This specification captured all the processes that would make up the system and the interactions over channels that would occur between them

2 A detailed description of each process that would be part of the system was made This description outlined the sequential computations that would be required to be processed so that the relevant information would be available to send on channels An input/output contract was developed for each sequential computation, i e given input information of a various format the sequential computation would guarantee output of a particular format

3 The various sequential computations were sorted into logical groupings and Java classes were created for each grouping The bulk of the processing that makes up the sequential computations was placed into these classes to minimise the complexity of the statements in the ϖ processes themselves

4 Once written the Java classes were unit tested to ensure that they met the input/output contract previously arrived at for them

5 A stripped down version of the ϖ system was written This stripped down version only contained communications code Following the implementation of this version of the system the ϖ code for it would be manually compared to the pi-calculus specification to informally ensure that the specification and implementation matched Ideally a formal process for the verification of implementation against specification would have been performed at this stage However the development of such a process fell outside the scope of this project and could therefore not be performed

6 The various sequential computation invocations required were added to the communications code These invocations were added at the points identified in the process analysis phase

7 The entire ϖ system would then be compiled Following the successful compilation of the ϖ system, it would then be deployed to a test environment and executed to ensure that the system behaved as expected

6 3 1 Reuse in ϖ systems

Once a ϖ system has been written aspects of it can be reused If identical functional requirements are made of processes in two separate systems then the same process can be used in both systems The process in question is simply written in a separate file and included in the compilation of both systems [4] Likewise if identical requirements are made of sequential computations these can be expressed as *CodeBlocks* and these can then be included in multiple ϖ systems

[4] The compilation of ϖ systems is detailed in the appendices

6.4 Conclusions

In the previous examples the majority of the functionality of the ϖ programming language have been used and demonstrated and the process for writing systems using the ϖ programming language has also been covered

Chapter 7

Conclusions

The goal of this research was to develop a programming language that was based on the π-calculus The nature of this programming language was intended to allow it to be used as a general purpose programming language although its primary use was to be in the implementation of large scale distributed systems

A number of objectives had to be completed in order to develop such a programming language The first was that a syntax and semantics that were similar to those of the π-calculus had to be devised The next objective, which was achieved in tandem with the first, was that a mechanism for performing complex sequential computations had to be integrated into the syntax and semantics of the programming language in such a manner as to allow the syntax and semantics of the programming language to still be reconcilable with those of the π-calculus

Another objective was that systems written in this programming language should be capable of being distributed with the minimum amount of effort and the maximum amount of transparency to the developer as possible This required the provision of mechanisms to enable distribution of systems at the lowest levels of the programming language By providing a high level of support and by doing it at a low level this objective was achieved

Following investigation into the π-calculus and existing implementations based on it a set of additional desired qualities for the new language were devised These additional properties became objectives in their own right that had to be satisfied by the language

The final, and arguable the most important, object was that the language devised should be simple to use and easy to understand This last objective had consequences for all aspects of the programming language, ranging from having to have a clear and concise syntax and semantics to ensuring that the compiler was easy to use and that the deployment process for systems written in this

language was simple to follow

Overall it is felt that these objectives were achieved and that a usable, computationally powerfully programming language that is capable of supporting large scale distributed systems was produced, and that this programming language was based on the π-calculus

7.1 Further work

While all the desired properties were incorporated into the language and while all the objectives, major and minor, of the language were completed it is felt that certain areas of the language could be expanded upon in further work

7 1 1 Sequential Computation notation

It could be desirable to extend the syntax and semantics of ϖ to include a notation that could be used in performing sequential computations in ϖ This could be beneficial as it may result in a simpler, and more controllable, mechanism for performing sequential computations and could also make the integration of the separate communications and computation aspects of ϖ cleaner and more elegant

7 1 2 Compiler support

As it stands compilation, debugging and deployment of ϖ systems must be done via the command line The integration of the ϖ language into a development environment would increase the ease with which the coding, compiling, debugging and deployment of ϖ could be achieved The development of such an environment, or the development of a series of plug-ins for an existing environment, would be a significant addition to ϖ

Appendix A

Building and using ϖ

A.1 Building the ϖ compiler and libraries

The source for the ϖ distribution is divided into three categories - the javacc code for the compiler, the Java code used by this compiler and the Java code which makes up the libraries that are used during run time To build the entire ϖ distribution all three categories of code must be built in different ways and the generated output files must then be bundled up into a jar file which will be the ϖ distribution [1]
To build the compiler and libraries the following steps can be followed or the included makefile can be used

1 Compile the java code used by the compiler (this and all other compilation steps should be performed from the root of the source directory)

   (a) javac varpi/helpers/* java

2 Compile the runtime libraries

   (a) javac varpi/imp/classServer/* java

   (b) javac varpi/imp/* java

   (c) rmic varpi imp iChannel

   (d) rmic varpi imp iTask

3 Compile the compiler

   (a) mkdir varpi/parser

[1] In order to build the ϖ distribution JDK1 4 x and Javacc 2 1 must be installed on the system

(b) javacc parser.jj

(c) javac varpi/parser/*.java

4 Prepare the jar file

(a) jar cvf varpi.jar varpi/imp/*.class varpi/imp/classServer/*.class varpi/helpers/*.class varpi/parser/*.class

A.2 Using the ϖ compiler/Building a ϖ system

1 (Optional) If any java classes have been written that will be used by the ϖ system being built then these classes must first be compiled

2 The varpi source files must be compiled In order to do this the names of these files are supplied to the ϖ compiler The only restriction place on the order in which these filenames are passed to the compiler is that the first file name must be that of the file that contains the ϖ System process

The usage of the ϖ compiler is

java -cp <varpi jar location> varpi parser Parser -sync <sname> -debug <filenames>

where

- *<varpi jar location>* is the location of the jar file built that represents the ϖ distribution
- *<sname>* is what the syncServer for the generated system should be called
- *<filenames>* are the filenames of all the ϖ source files

When using the ϖ compiler the *-debug* switch is optional and its use simply results in additional debug information being generated and included in the system which will be displayed at runtime

3 The successful compilation of a ϖ system results in the generation of a number of java classes The final compulsory step in the compilation of a ϖ system involves the compilation of these generated java files In order for this compilation to success the jar file built for the ϖ distribution and any external java classes required must be included in the classpath The compilation is performed by typing *javac * java* from the directory from the working directory

4 (Optional) If desired the class files resulting from the compilation of the generated java files can be packaged into a jar file for ease of distribution This step is recommended

A.3 Running a ϖ system

1 **Distribute required files**

In order to run the ϖ system on numerous distributed hosts it is necessary for the libraries for both the ϖ distribution and for the system in question to be present on each of the machines that will form part of the system This distribution is done "out of band"

2 **Rmiregistry**

As the ϖ distribution makes use of Java RMI remote objects it is required that an instance of the rmiregistry is running on each host machine Additionally the location of the jar file for the ϖ distribution must be in the classpath for the rmiregistry

3 **SyncServer running**

Each ϖ system requires a ϖ syncServer to be running The compiler of a ϖ system results in the generation of a tailored syncServer This generated syncServer has the sname name as that which is supplied to the ϖ compiled via the *-syncServer* switch The invocation of a syncServer is a simple process as syncServer only requires a single argument - the number of a free port on the host machine

Usage *java -cp <location of ϖ jar> <syncServer name> <port>*

where

- *<location of ϖ jar>* is the location of the jar file for the ϖ distribution
- *<syncServer>* is the name of the syncServer class
- *<port>* is the number of a free port on the host machine It is also required that the port immediately above this port is also free

4 **Top level processes started**

A ϖ system, when compiled, will generated a number of java classes which contain a public stat void main method One such class is generated for each top level process in the system These generated executable classes are named the same as the top level processes but with the word "Starter" attached to the end of the name

In order for the execution of a ϖ system to commence each top level process, bar replicated top-level processes, must first be started The manner of doing so is identical for each top level process

java -cp <classpath> <tlpStarter> <port> <jar> <ssip> <ssport> <local ip >

where

- *<classpath>* is the classpath required to run the program, it must include the location of the ϖ distribution jar, the jar file for this system, and any external java classes required
- *<tlpStarter>* is the name of the executable class file to run (top level process name plus the string "Starter"
- *<port>* is the number of a free port on the host machine
- *<jar>* is the location of the ϖ distribution jar file
- *<ssip>* is the ip address of the sync server
- *<ssport>* is the number of the port on which the syncServer is listening
- *<local ip >* is the local ip address

By starting the rmiregistry and each of the top-level processes as detailed above the execution of the ϖ system in question should begin Termination of the system is handled automatically by the system

Appendix B

Example 1 code

B.1 ϖ code

```
1 {
2   pki *,
3   java security cert *,
4   java security *,
5   javax crypto spec *,
6   javax crypto *,
7 }
8
9 System Sys
10 {
11   Channel a
12
13   (CertAuth(a)|!ClientCreateCert(a))
14 }
15
16 Process CertAuth(cert)
17 {
18   <getInfo>()(filename,passphrase)
19   <getIssuer>(filename,passphrase)(i)
20
21   (!Issuer(cert,i))
22 }
23
24 Process Issuer(in, issuer)
25 {
26   in(channel)
27   channel(certRequest)
28   <issueCert>(issuer,certRequest)(cert)
29   channel<cert>
30   0
31 }
32
```

```
33 Code issueCert(issuer,request)(cert)
34 {
35   /&
36   CertIssuer i = (CertIssuer)issuer,
37   byte[] req = (byte[]) request,
38
39   cert = i processCertificateRequest(req),
40   &/
41 }
42
43 Code getIssuer(fn,pp)(issuer)
44 {
45   /&
46   CA theCA = new CA((String)fn,(String)pp),
47   CertIssuer i = theCA createIssuer(),
48
49   issuer = i,
50   &/
51 }
52
53 Code getInfo()(fn,pp)
54 {
55   /&
56   try
57   {
58     LineNumberReader lnr =
59       new LineNumberReader(new InputStreamReader(System in)),
60     System out println("Enter the ca name"),
61     fn = lnr readLine(),
62     System out println("Enter the passphrase"),
63     pp = lnr readLine(),
64   }
65   catch(Exception e)
66   {
67     e printStackTrace(),
68   }
69   &/
70 }
71
72
73 Process ClientCreateCert(chan)
74 {
75   <loadCACert>()(cacert)
76   <createCertificateAndRequest>()(client,req)
77
78   Channel tmp
79   chan<tmp>
80
81   tmp<req>
82
```

```
83  tmp(cert)
84
85  <storeClient>(cert,client,cacert)()
86  0
87 }
88
89 Code loadCACert()(cert)
90 {
91  /&
92  String filename = Client getCAFileName(),
93  byte[] cert_bytes = Client loadCACert(filename),
94  cert = cert_bytes,
95  &/
96 }
97
98
99 Code createCertificateAndRequest()(client,req)
100 {
101  /&
102  Client c = new Client(),
103  byte[] name = Client getName(),
104
105  byte[] tmp = c generateCertificateRequest(name),
106
107  client = c,
108  req = tmp,
109  &/
110 }
111
112 Code storeClient(cert,client,cacert)()
113 {
114  /&
115  byte[] theCACert = (byte[])cacert,
116
117  byte[] theCert = (byte[])cert,
118  Client c = (Client)client,
119
120  c setCertificate(theCert,theCACert),
121
122  String pp = Client getPassPhrase(),
123  String fn = Client getFilename(),
124
125  c store(pp,fn),
126  &/
127 }
```

B 2 Java code

B 2 1 CA java

```
1 package pki;
2
3 import pki.*;
4
5 import java.security.cert.*;
6 import java.security.*;
7 import javax.crypto.spec.*;
8 import javax.crypto.*;
9 import java.io.*;
10 import iaik.pkcs.pkcs10.CertificateRequest;
11 import iaik.asn1.structures.*;
12 import iaik.asn1.*;
13
14 import java.util.*;
15 import java.math.*;
16
17
18 public class CA implements Serializable
19 {
20   private KeyPair m_keys;
21   private X509Certificate m_cert;
22   private String m_filename;
23
24   public CA(String filename,String pp)
25   {
26     m_filename = filename;
27     loadInfo(pp);
28   }
29
30   public PublicKey getPublic()
31   {
32     return m_keys.getPublic();
33   }
34
35   public X509Certificate getCert()
36   {
37     return m_cert;
38   }
39
40   public void loadInfo(String pp)
41   {
42     try
43     {
44       FileInputStream fis =
45         new FileInputStream(m_filename + ".info");
46
47       int b = fis.read();
```

```
48
49      ByteArrayOutputStream baos = new ByteArrayOutputStream(),
50
51      while( b != -1)
52      {
53        baos write(b),
54        b= fis read(),
55      }
56      //DECRYPT
57      byte[] iv_bytes = "this is the iv" getBytes(),
58      SecretKeySpec sks = new SecretKeySpec(pp getBytes(), 0, 8, "DES"),
59      IvParameterSpec ap = new IvParameterSpec(iv_bytes, 0, 8 ),
60
61      Cipher c = Cipher getInstance("DES/CBC/PKCS5Padding"),
62      SecureRandom sr = new SecureRandom("This is a very bad seed" getBytes()),
63      c init(Cipher DECRYPT_MODE, sks, ap, sr),
64
65      byte[] final_bytes = c doFinal(baos toByteArray()),
66
67      ByteArrayInputStream bais = new ByteArrayInputStream(final_bytes),
68      ObjectInputStream ois = new ObjectInputStream(bais),
69
70      m_keys = (KeyPair)ois readObject(),
71      m_cert = (X509Certificate)ois readObject(),
72
73      }
74      catch(Exception e)
75      {
76        e printStackTrace(),
77      }
78  }
79
80  public static void generateAndStore(String passphrase, String filename)
81  {
82    try
83    {
84      String seed = new String(),
85      seed += System currentTimeMillis(),
86
87      SecureRandom sec_random = new SecureRandom(seed getBytes()),
88
89      KeyPairGenerator key_gen = KeyPairGenerator getInstance("RSA"),
90      key_gen initialize(512, sec_random),
91      KeyPair key_pair = key_gen generateKeyPair(),
92
93      Name n = new Name(),
94      n addRDN(ObjectID country, "IE"),
95      n addRDN(ObjectID locality, "DUBLIN"),
96      n addRDN(ObjectID organization ,"DCU"),
97      n addRDN(ObjectID organizationalUnit ,"PG"),
```

```
98      n addRDN(ObjectID commonName ,"CA"),
99
100     iaik x509 X509Certificate cert = new iaik x509 X509Certificate(),
101     cert setIssuerDN(n),
102     cert setSubjectDN(n),
103     cert setPublicKey(key_pair getPublic()),
104     cert setSerialNumber(new BigInteger("000000000001")),
105     GregorianCalendar date = (GregorianCalendar)Calendar getInstance(),
106
107     date add(Calendar MONTH, -1),
108     cert setValidNotBefore(date getTime()),
109     date add(Calendar MONTH, 5),
110     cert setValidNotAfter(date getTime()),
111     cert sign(AlgorithmID sha1WithRSAEncryption, key_pair getPrivate()),
112
113     ByteArrayOutputStream baos = new ByteArrayOutputStream(),
114     ObjectOutputStream oos = new ObjectOutputStream(baos),
115
116     oos writeObject(key_pair),
117     oos writeObject(cert),
118     System out println(cert getClass() getName()),
119     oos close(),
120
121     byte[] iv_bytes = "this is the iv" getBytes(),
122     SecretKeySpec sks = new SecretKeySpec(passphrase getBytes(), 0, 8, "DES"),
123     IvParameterSpec ap = new IvParameterSpec(iv_bytes, 0, 8 ),
124
125     Cipher c = Cipher getInstance("DES/CBC/PKCS5Padding"),
126     SecureRandom sr = new SecureRandom("This is a very bad seed" getBytes()),
127     c init(Cipher ENCRYPT_MODE, sks, ap, sr),
128
129     byte[] final_bytes = c doFinal(baos toByteArray()),
130
131     FileOutputStream fos = new FileOutputStream(filename + " info" ),
132     fos write(final_bytes),
133     fos close(),
134
135     fos = new FileOutputStream(filename + " crt"),
136     fos write(cert toByteArray()),
137     fos close(),
138
139     File f = new File(filename + " crl"),
140     f createNewFile(),
141
142   }
143   catch(Exception e)
144   {
145     e printStackTrace(),
146   }
147 }
```

```
148
149  public static void printCAInfo(String passphrase, String filename)
150  {
151    try
152    {
153      FileInputStream fis = new FileInputStream(filename + " info"),
154
155      int b = fis read(),
156
157      ByteArrayOutputStream baos = new ByteArrayOutputStream(),
158
159      while( b != -1)
160      {
161        baos write(b),
162        b= fis read(),
163      }
164      //DECRYPT
165      byte[] iv_bytes = "this is the iv" getBytes(),
166      SecretKeySpec sks = new SecretKeySpec(passphrase getBytes(), 0, 8, "DES"),
167      IvParameterSpec ap = new IvParameterSpec(iv_bytes, 0, 8 ),
168
169      Cipher c = Cipher getInstance("DES/CBC/PKCS5Padding"),
170      SecureRandom sr = new SecureRandom("This is a very bad seed" getBytes()),
171      c init(Cipher DECRYPT_MODE, sks, ap, sr),
172
173      byte[] final_bytes = c doFinal(baos toByteArray()),
174
175      ByteArrayInputStream bais = new ByteArrayInputStream(final_bytes),
176      ObjectInputStream ois = new ObjectInputStream(bais),
177
178      KeyPair kp = (KeyPair)ois readObject(),
179      Object cert = ois readObject(),
180      System out println(cert getClass() getName()),
181
182      System out println("---------------------"),
183      System out println(cert),
184      System out println("---------------------"),
185    }
186    catch(Exception e)
187    {
188      e printStackTrace(),
189    }
190  }
191
192  public CertIssuer createIssuer()
193  {
194    return new CertIssuer(m_keys, m_cert),
195  }
196
197  public static void main(String args[])
```

```
198   {
199     CA generateAndStore(args[0], args[1]),
200     CA printCAInfo(args[0], args[1]),
201   }
202 }
```

B 2 2 CertIssuer java

```
1
2 package pki;
3
4 import pki.*;
5
6 import java.security.*;
7 import javax.crypto.spec.*;
8 import javax.crypto.*;
9 import java.io.*;
10 import iaik.pkcs.pkcs10.CertificateRequest;
11 import iaik.asn1.structures.*;
12 import iaik.asn1.*;
13 import java.security.cert.*;
14
15 import java.util.*;
16 import java.math.*;
17
18 public class CertIssuer implements Serializable
19 {
20   private KeyPair m_keys;
21   private X509Certificate m_cert;
22
23   public CertIssuer(KeyPair keys, X509Certificate cert)
24   {
25     m_keys = keys;
26     m_cert = cert;
27   }
28
29   public PublicKey getPublic()
30   {
31     return m_keys.getPublic();
32   }
33
34   public X509Certificate getCert()
35   {
36     return m_cert;
37   }
38
39   public byte[] processCertificateRequest(byte[] request)
40   {
41     try
42     {
43       CertificateRequest cert_request = new CertificateRequest(request);
44
45       Name subject = cert_request.getSubject();
46       PublicKey pk = cert_request.getPublicKey();
47
48       String l = new String();
49       l += System.currentTimeMillis();
```

```
50
51      SecureRandom sr = new SecureRandom(l getBytes()),
52      String serial = new String(),
53      serial += sr nextLong(),
54
55      iaik x509 X509Certificate t_cert
56        = new iaik x509 X509Certificate(m_cert getEncoded()),
57      iaik asn1 structures Name issuer
58        = (iaik asn1 structures Name)t_cert getSubjectDN(),
59
60      iaik x509 X509Certificate cert = new iaik x509 X509Certificate(),
61      cert setIssuerDN(issuer),
62      cert setSubjectDN(subject),
63      cert setPublicKey(pk),
64      cert setSerialNumber(new BigInteger(serial)),
65      GregorianCalendar date = (GregorianCalendar)Calendar getInstance(),
66      date add(Calendar MONTH, -1),
67      cert setValidNotBefore(date getTime()),
68      date add(Calendar MONTH, 5),
69      cert setValidNotAfter(date getTime()),
70      cert sign(AlgorithmID sha1WithRSAEncryption, m_keys getPrivate()),
71
72      return cert toByteArray(),
73    }
74    catch(Exception e)
75    {
76      e printStackTrace(),
77    }
78    return null,
79  }
80}
```

B 2 3 Client java

```
1 package pki;
2
3 import pki *;
4
5 import java security *;
6 import javax crypto spec *;
7 import javax crypto *;
8 import java io *;
9 import iaik pkcs pkcs10 CertificateRequest;
10 import iaik asn1 structures *;
11 import iaik asn1 *;
12 import java security cert *;
13
14 public class Client implements Serializable
15 {
16   private KeyPair m_keys = null;
17   private X509Certificate m_cert = null;
18   private X509Certificate ca_cert = null;
19
20   public Client()
21   {
22   //Case 1
23   }
24
25   public Client(String passphrase, String filename, String cafilename)
26   {
27     try
28     {
29       FileInputStream fis = new FileInputStream(filename + " crt");
30       ByteArrayOutputStream bt = new ByteArrayOutputStream();
31
32       int b = fis read();
33
34       while( b != -1)
35       {
36         bt write(b);
37         b = fis read();
38       }
39
40       fis close();
41       bt close();
42
43       byte[] cert_bytes = bt toByteArray();
44       fis = new FileInputStream(cafilename +" crt");
45       bt = new ByteArrayOutputStream();
46
47       b = fis read();
48
49       while( b != -1)
```

```
50      {
51        bt write(b),
52        b = fis read(),
53      }
54
55      fis close(),
56      bt close(),
57
58      byte[] cacert_bytes = bt toByteArray(),
59      fis = new FileInputStream(filename + " key"),
60      bt = new ByteArrayOutputStream(),
61
62      b = fis read(),
63
64      while( b != -1)
65      {
66        bt write(b),
67        b = fis read(),
68      }
69
70      byte[] iv_bytes = "this is the iv" getBytes(),
71      SecretKeySpec sks = new SecretKeySpec(passphrase getBytes(), 0, 8, "DES"),
72      IvParameterSpec ap = new IvParameterSpec(iv_bytes, 0, 8 ),
73
74      Cipher c = Cipher getInstance("DES/CBC/PKCS5Padding"),
75      SecureRandom sr = new SecureRandom("This is a very bad seed" getBytes()),
76      c init(Cipher DECRYPT_MODE, sks, ap, sr),
77
78      byte[] final_bytes = c doFinal(bt toByteArray()),
79
80      ByteArrayInputStream bais = new ByteArrayInputStream(final_bytes),
81      ObjectInputStream ois = new ObjectInputStream(bais),
82
83      KeyPair kp = (KeyPair)ois readObject(),
84      CertificateFactory cf = CertificateFactory getInstance("X509"),
85      ByteArrayInputStream cbais = new ByteArrayInputStream(cert_bytes),
86      X509Certificate cert = (X509Certificate)cf generateCertificate(cbais),
87
88      cbais = new ByteArrayInputStream(cacert_bytes),
89      X509Certificate cacert = (X509Certificate)cf generateCertificate(cbais),
90
91      cert verify(cacert getPublicKey()),
92      m_cert = cert,
93      m_keys = kp,
94      ca_cert = cacert,
95
96    }
97    catch(Exception e)
98    {
99      e printStackTrace(),
```

```
100     }
101   }
102
103   public Client(byte[] certBytes, byte[] keypair, String passphrase)
104   {
105     try
106     {
107       byte[] iv_bytes = "this is the iv" getBytes(),
108       SecretKeySpec sks = new SecretKeySpec(passphrase getBytes(), 0, 8, "DES"),
109       IvParameterSpec ap = new IvParameterSpec(iv_bytes, 0, 8 ),
110
111       Cipher c = Cipher getInstance("DES/CBC/PKCS5Padding"),
112       SecureRandom sr = new SecureRandom("This is a very bad seed" getBytes()),
113       c init(Cipher DECRYPT_MODE, sks, ap, sr),
114
115       byte[] final_bytes = c doFinal(keypair),
116
117       ByteArrayInputStream bais = new ByteArrayInputStream(final_bytes),
118       ObjectInputStream ois = new ObjectInputStream(bais),
119
120       KeyPair kp = (KeyPair)ois readObject(),
121       CertificateFactory cf = CertificateFactory getInstance("X509"),
122       ByteArrayInputStream cbais = new ByteArrayInputStream(certBytes),
123       X509Certificate cert = (X509Certificate)cf generateCertificate(cbais),
124
125       m_cert = cert,
126       m_keys = kp,
127     }
128     catch(Exception e)
129     {
130       e printStackTrace(),
131     }
132   }
133
134   public KeyPair generateKeyPair(int len)
135   {
136     try
137     {
138       String seed = new String(),
139       seed += System currentTimeMillis(),
140       SecureRandom sec_random = new SecureRandom(seed getBytes()),
141
142       KeyPairGenerator key_gen = KeyPairGenerator getInstance("RSA"),
143       key_gen initialize(len, sec_random),
144       m_keys = key_gen generateKeyPair(),
145
146     }
147     catch(Exception e)
148     {
149       e printStackTrace()
```

```
150     }
151
152     return m_keys;
153   }
154
155   public byte[] generateCertificateRequest(byte[] name)
156   {
157     try
158     {
159       if(m_keys == null)
160       {
161         generateKeyPair(512);
162       }
163       Name n = new Name(name);
164       CertificateRequest c = new CertificateRequest(m_keys.getPublic(),n);
165       c.sign(AlgorithmID.sha1WithRSAEncryption,m_keys.getPrivate());
166
167       byte[] bytes = c.toByteArray();
168
169       return bytes;
170     }
171     catch(Exception e)
172     {
173       e.printStackTrace();
174       return null;
175     }
176   }
177
178   public void setCertificate(byte[] c, byte[] ca)
179   {
180     try
181     {
182       CertificateFactory cf = CertificateFactory.getInstance("X509");
183       ByteArrayInputStream bais = new ByteArrayInputStream(c);
184       X509Certificate cert = (X509Certificate)cf.generateCertificate(bais);
185
186       bais = new ByteArrayInputStream(ca);
187       X509Certificate cacert = (X509Certificate)cf.generateCertificate(bais);
188
189       setCertificate(cert,cacert);
190     }
191     catch(Exception e)
192     {
193       e.printStackTrace();
194     }
195   }
196
197   public void setCertificate(X509Certificate c, X509Certificate ca) throws Exception
198   {
199     c.verify(ca.getPublicKey());
```

```
200
201     if( c getPublicKey() equals(m_keys getPublic()))
202     {
203       m_cert = c,
204     }
205     else
206     {
207       throw new Exception("Invalid cert"),
208     }
209   }
210
211   public void store(String passphrase, String prefix)
212   {
213     try
214     {
215
216       if(m_keys != null)
217       {
218         ByteArrayOutputStream baos = new ByteArrayOutputStream(),
219         ObjectOutputStream oos = new ObjectOutputStream(baos),
220
221         oos writeObject(m_keys),
222         oos close(),
223
224         SecretKeySpec new_sks = new SecretKeySpec( passphrase getBytes(), 0, 8, "DES"),
225         SecureRandom sr = new SecureRandom("this is a very bad seed" getBytes()),
226         byte[] iv_bytes = "this is the iv" getBytes(),
227
228         IvParameterSpec ap = new IvParameterSpec(iv_bytes,0,8),
229         Cipher c  = Cipher getInstance("DES/CBC/PKCS5Padding"),
230         c init(Cipher ENCRYPT_MODE, new_sks, ap, sr),
231         byte[] bytes = c doFinal(baos toByteArray()),
232
233
234         FileOutputStream fos = new FileOutputStream( new String( prefix + " key")),
235         fos write(bytes),
236        fos close(),
237       }
238
239       if( m_cert != null)
240       {
241         FileOutputStream fos = new FileOutputStream( new String( prefix + " crt")),
242         fos write(m_cert getEncoded()),
243         fos close(),
244       }
245     }
246     catch(Exception e)
247     {
248       e printStackTrace(),
249     }
```

```
250   }
251
252   public X509Certificate getCert()
253   {
254     return m_cert;
255   }
256
257   public KeyPair getKeys()
258   {
259     return m_keys;
260   }
261
262   public static byte[] loadCACert(String filename)
263   {
264     try
265     {
266       FileInputStream fis = new FileInputStream(filename + ".crt");
267       ByteArrayOutputStream bt = new ByteArrayOutputStream();
268
269       int b = fis.read();
270
271       while( b != -1)
272       {
273         bt.write(b);
274         b = fis.read();
275       }
276
277       fis.close();
278       bt.close();
279
280       byte[] cert_bytes = bt.toByteArray();
281
282       return cert_bytes;
283     }
284     catch(Exception e)
285     {
286       e.printStackTrace();
287       return null;
288     }
289   }
290
291   public static String getCAFileName()
292   {
293     try
294     {
295       LineNumberReader lnr =
296         new LineNumberReader(new InputStreamReader(System.in));
297       System.out.println("Enter the location of the CA certificate ");
298       String cacertfilename = lnr.readLine();
299
```

```
300       return cacertfilename,
301     }
302     catch(Exception e)
303     {
304       e printStackTrace(),
305       return null,
306     }
307   }
308
309   public static byte[] getName()
310   {
311     try
312     {
313       LineNumberReader lnr =
314          new LineNumberReader(new InputStreamReader(System in)),
315
316       System out println("Enter country code  e g  IE"),
317       String cc = lnr readLine(),
318       System out println("Enter locality  e g  Dublin"),
319       String loc = lnr readLine(),
320       System out println("Enter organization  e g  DCU"),
321       String org = lnr readLine(),
322       System out println("Enter organizational unit  e g  POSTGRAD"),
323       String unit = lnr readLine(),
324       System out println("Enter common name  e g  John Doe"),
325       String cn = lnr readLine(),
326
327       Name name = new Name(),
328       name addRDN(ObjectID country, cc),
329       name addRDN(ObjectID locality, loc),
330       name addRDN(ObjectID organization ,org),
331       name addRDN(ObjectID organizationalUnit ,unit),
332       name addRDN(ObjectID commonName ,cn),
333
334       return name getEncoded(),
335     }
336     catch(Exception e)
337     {
338       e printStackTrace(),
339     }
340     return null,
341   }
342
343   public static String getPassPhrase()
344   {
345     try
346     {
347       LineNumberReader lnr =
348         new LineNumberReader(new InputStreamReader(System in)),
349
```

```
350       System out println("Enter your passphrase "),
351       String pp = lnr readLine(),
352
353       return pp,
354     }
355     catch(Exception e)
356     {
357       return null,
358     }
359   }
360
361   public static String getFilename()
362   {
363     try
364     {
365       LineNumberReader lnr =
366         new LineNumberReader(new InputStreamReader(System in)),
367
368
369       System out println("Enter the prefix for all client files "),
370       String pp = lnr readLine(),
371
372       return pp,
373     }
374     catch(Exception e)
375     {
376       return null,
377     }
378   }
379 }
```

Appendix C

Example 2 code

C.1 ϖ code

C.2 ϖ code

```
1 {
2 pki *,
3 java security cert *,
4 java security *,
5 javax crypto spec *,
6 javax crypto *,
7 }
8
9 System Sys
10 {
11   Channel a,b,c,w
12
13   (CertAuth(a,b,c)|'ClientCreateCert(a)
14     |ServiceProvider(b,w)|'ClientServiceRequest(c,w))
15 }
16
17 Process CertAuth(cert,spCert,spCertOut)
18 {
19   <getInfo>()(filename,passphrase)
20   <createCA>(filename,passphrase)(ca)
21   <getIssuer>(ca)(i)
22
23   ('Issuer(cert,i)|ServiceProviderIssuer(spCert,i,spCertOut))
24 }
25
26 Process Issuer(in, issuer)
27 {
28   in(channel)
29
```

```
30   channel(certRequest)
31
32   <issueCert>(issuer,certRequest)(cert)
33
34   channel<cert>
35   0
36 }
37
38 Process ServiceProviderIssuer(in, issuer,out)
39 {
40   in(channel)
41
42   channel(certRequest)
43
44   <issueCert>(issuer,certRequest)(cert)
45
46   channel<cert>
47   ('DistributeCert(out,cert))
48 }
49
50 Process DistributeCert(out,cert)
51 {
52   out<cert>
53   0
54 }
55
56 Process ServiceProvider(chan,work)
57 {
58   <loadCACert>()(cacert)
59
60   <createCertificateAndRequestSP>(cacert)(sp,req)
61
62   Channel tmp
63   chan<tmp>
64
65   tmp<req>
66
67   tmp(cert)
68
69   <setCertificate>(cert,sp,cacert)(newSP)
70   ('Servicer(work,newSP))
71 }
72
73 Process Servicer(work,self)
74 {
75   work(channel)
76   channel(clientCert)
77   channel(pac1)
78
79   <processClientRequest>(clientCert,pac1,self)(randA)
```

```
80
81  <createServiceResponse>(clientCert,self,randA)(pac2,randB)
82
83  channel<pac2>
84
85  channel(res)
86
87  <processClientResponse>(clientCert,self,res,randB)(encKey,key)
88
89  channel<encKey>
90
91  Channel service1,service2
92
93  channel<service1>
94  channel<service2>
95
96  +(service1(a)(Service1(a)))+(service2(b)(Service2(b)))
97}
98
99 Process Service1(in)
100{
101  <createResponse1>()(res)
102  in<res>
103  0
104}
105
106 Process Service2(in)
107{
108  <createResponse1>()(res)
109  in<res>
110  0
111}
112
113 Process ClientCreateCert(chan)
114{
115  <loadCACert>()(cacert)
116
117  <createCertificateAndRequest>()(client,req)
118
119  Channel tmp
120  chan<tmp>
121
122  tmp<req>
123
124  tmp(cert)
125
126  <setCertificate>(cert,client,cacert)(newClient)
127  <storeClient>(newClient)()
128
129  0
```

```
130 }
131
132 Process ClientServiceRequest(cert, work)
133 {
134   <loadCACert>()(cacert)
135   <loadSelf>(cacert)(self)
136
137   cert(spCert)
138
139   <verifyCertIssuer>(cacert,spCert)()
140
141   //----------------Start Protocol
142   <createClientRequest>(spCert,self)(packet1, randA)
143
144   Channel chan
145
146   work<chan>
147
148   <getOwnCert>(self)(ownCert)
149   chan<ownCert>
150   chan<packet1>
151
152   chan(packet2)
153
154   <processServiceResponse>(packet2,spCert,self,randA)(packet3)
155
156   chan<packet3>
157
158   chan(encKey)
159
160   chan(service1)
161   chan(service2)
162
163   <extractKey>(spCert,encKey,self)(key)
164
165   <whichService>(service1,service2)(service)
166
167   Channel chan
168
169   service<chan>
170
171   chan(resp)
172
173   </&System out println((String)resp),&/>(resp)()
174   0
175 }
176
177 Code whichService(s1,s2)(s)
178 {
179   /&
```

```
180   if(Client chooseFirst())
181   s = s1,
182   else
183   s = s2,
184   &/
185 }
186
187 Code extractKey(spCert,encKey,s)(k)
188 {
189   /&
190   Client c = (Client)s,
191
192   byte[] ks = c extractKey((byte[])spCert,(byte[])encKey),
193
194   k = ks,
195   &/
196 }
197
198 Code processServiceResponse(pac,cert,self, rand)(returnPac)
199 {
200   /&
201   Client c = (Client)self,
202
203   returnPac = c createClientResponse((byte[])cert,(byte[])pac,(Long)rand),
204   &/
205 }
206
207 Code getOwnCert(self)(cert)
208 {
209   /&
210   PkiBase base = (PkiBase)self,
211
212   cert = base getCertBytes(),
213   &/
214 }
215
216 Code createClientRequest(provider,self)(packet, rand)
217 {
218   /&
219   Client c = (Client)self,
220   Long l = new Long(System currentTimeMillis()),
221
222   packet = c createRequest((byte[])provider,l),
223   rand = l,
224   &/
225 }
226
227 Code verifyCertIssuer(cacert,spCert)()
228 {
229   /&
```

```
230   Client verifyCertIssuer((byte[])cacert,(byte[])spCert),
231   &/
232 }
233
234 Code loadSelf(cacert)(self)
235 {
236   /&
237   String pp = Client getPassPhrase(),
238   String fn = Client getFilename(),
239
240   Client c = new Client(pp,fn,(byte[])cacert),
241
242   self = c,
243   &/
244 }
245
246 Code issueCert(issuer,request)(cert)
247 {
248   /&
249   CertIssuer i = (CertIssuer)issuer,
250   byte[] req = (byte[]) request,
251
252   cert = i processCertificateRequest(req),
253   &/
254 }
255
256 Code issureCert(issuer,req)(cert)
257 {
258   /&
259   CertIssuer i = (CertIssuer)issuer,
260   byte[] the_request = (byte[])req,
261
262   cert = processCertificateRequest(the_request),
263   &/
264 }
265
266 Code getIssuer(ca)(issuer)
267 {
268   /&
269   CA theCA = (CA)ca,
270   CertIssuer i = theCA createIssuer(),
271
272   issuer = i,
273   &/
274 }
275
276 Code createCA(fn,pp)(ca)
277 {
278   /&
279   CA theCA = new CA((String)fn,(String)pp),
```

```
280   ca = theCA,
281   &/
282 }
283
284 Code getInfo()(fn,pp)
285 {
286   /&
287   try
288   {
289     LineNumberReader lnr =
290       new LineNumberReader(new InputStreamReader(System in)),
291     System out println("Enter the ca name"),
292     fn = lnr readLine(),
293     System out println("Enter the passphrase"),
294     pp = lnr readLine(),
295   }
296   catch(Exception e)
297   {
298     e printStackTrace(),
299   }
300   &/
301 }
302
303 Code createResponse1()(res)
304 {
305   /&
306   res = new String("RES1"),
307   &/
308 }
309
310 Code createResponse2()(res)
311 {
312   /&
313   res = new String("RES2"),
314   &/
315 }
316
317 Code processClientResponse(cert,self,pac,rand)(ekey,key)
318 {
319   /&
320   SP sp = (SP)self,
321
322   byte[] keyBytes = sp processClientResponse((byte[])cert,(byte[])pac,(Long)rand),
323
324   ekey = sp encryptKeyBytes((byte[])cert,keyBytes),
325   key = keyBytes,
326   &/
327 }
328
329 Code createServiceResponse(cc,s,rA)(p,rB)
```

```
330 {
331   /&
332   SP sp = (SP)s,
333   Long l = new Long(System currentTimeMillis()),
334
335   p = sp createServiceResponse((Long)rA,l,(byte[])cc),
336   rB = l,
337   &/
338 }
339
340 Code processClientRequest(clientCert,packet,self)(randA)
341 {
342   /&
343   SP sp = (SP)self,
344   randA = sp processClientRequest((byte[])packet,(byte[])clientCert),
345   &/
346 }
347
348 Code createCertificateAndRequestSP(ca)(ret,req)
349 {
350   /&
351   SP sp = new SP((byte[])ca),
352
353   byte[] tmp = sp generateCertificateRequest(),
354
355   ret = sp,
356   req = tmp,
357   &/
358 }
359
360 Code loadCACert()(cert)
361 {
362   /&
363   String filename = Client getCAFileName(),
364   byte[] cert_bytes = Client loadCACert(filename),
365   cert = cert_bytes,
366   &/
367 }
368
369
370 Code createCertificateAndRequest()(client,req)
371 {
372   /&
373   Client c = new Client(),
374   byte[] name = Client getName(),
375
376   byte[] tmp = c generateCertificateRequest(name),
377
378   client = c,
379   req = tmp,
```

```
}
```

```
380   &/
381 }
382
383 Code setCertificate(cert,client,cacert)(newClient)
384 {
385   /&
386   try
387   {
388     byte[] theCACert = (byte[])cacert,
389
390     byte[] theCert = (byte[])cert,
391     PkiBase c = (PkiBase)client,
392
393     c setCertificate(theCert,theCACert),
394
395     newClient = c,
396   }
397   catch(Exception e)
398   {
399     e printStackTrace(),
400   }
401   &/
402 }
403
404 Code storeClient(client)()
405 {
406   /&
407   Client c = (Client)client,
408
409   String pp = Client getPassPhrase(),
410   String fn = Client getFilename(),
411
412   c store(pp,fn),
413   &/
414 }
```

C.3 Java code

C.3.1 CA.java

```
1 package pki;
2
3 import pki.*;
4
5 import java.security.cert.*;
6 import java.security.*;
7 import javax.crypto.spec.*;
8 import javax.crypto.*;
9 import java.io.*;
10 import iaik.pkcs.pkcs10.CertificateRequest;
11 import iaik.asn1.structures.*;
12 import iaik.asn1.*;
13
14 import java.util.*;
15 import java.math.*;
16
17
18 public class CA implements Serializable
19 {
20   private KeyPair m_keys;
21   private X509Certificate m_cert;
22   private String m_filename;
23
24   public CA(String filename,String pp)
25   {
26     m_filename = filename;
27     loadInfo(pp);
28   }
29
30   public PublicKey getPublic()
31   {
32     return m_keys.getPublic();
33   }
34
35   public X509Certificate getCert()
36   {
37     return m_cert;
38   }
39
40   public void loadInfo(String pp)
41   {
42     try
43     {
44       FileInputStream fis =
45         new FileInputStream(m_filename + ".info");
46
47       int b = fis.read()
```

```
48
49      ByteArrayOutputStream baos = new ByteArrayOutputStream();
50
51      while( b != -1)
52      {
53        baos write(b);
54        b= fis read();
55      }
56      //DECRYPT
57      byte[] iv_bytes = "this is the iv" getBytes();
58      SecretKeySpec sks = new SecretKeySpec(pp getBytes(), 0, 8, "DES");
59      IvParameterSpec ap = new IvParameterSpec(iv_bytes, 0, 8 );
60
61      Cipher c = Cipher getInstance("DES/CBC/PKCS5Padding");
62      SecureRandom sr = new SecureRandom("This is a very bad seed" getBytes());
63      c init(Cipher DECRYPT_MODE, sks, ap, sr);
64
65      byte[] final_bytes = c doFinal(baos toByteArray());
66
67      ByteArrayInputStream bais = new ByteArrayInputStream(final_bytes);
68      ObjectInputStream ois = new ObjectInputStream(bais);
69
70      m_keys = (KeyPair)ois readObject();
71      m_cert = (X509Certificate)ois readObject();
72    }
73    catch(Exception e)
74    {
75      e printStackTrace();
76    }
77  }
78
79  public static void generateAndStore(String passphrase, String filename)
80  {
81    try
82    {
83      String seed = new String();
84      seed += System currentTimeMillis();
85
86      SecureRandom sec_random = new SecureRandom(seed getBytes());
87
88      KeyPairGenerator key_gen = KeyPairGenerator getInstance("RSA");
89      key_gen initialize(512, sec_random);
90      KeyPair key_pair = key_gen generateKeyPair();
91
92      Name n = new Name();
93      n addRDN(ObjectID country, "IE");
94      n addRDN(ObjectID locality, "DUBLIN");
95      n addRDN(ObjectID organization ,"DCU");
96      n addRDN(ObjectID organizationalUnit ,"PG");
97      n addRDN(ObjectID commonName ,"CA");
```

```
98
99
100     iaik.x509.X509Certificate cert = new iaik.x509.X509Certificate();
101     cert.setIssuerDN(n);
102     cert.setSubjectDN(n);
103     cert.setPublicKey(key_pair.getPublic());
104     cert.setSerialNumber(new BigInteger("000000000001"));
105     GregorianCalendar date = (GregorianCalendar)Calendar.getInstance();
106
107     date.add(Calendar.MONTH, -1);
108     cert.setValidNotBefore(date.getTime());
109     date.add(Calendar.MONTH, 5);
110     cert.setValidNotAfter(date.getTime());
111     cert.sign(AlgorithmID.sha1WithRSAEncryption, key_pair.getPrivate());
112
113
114     ByteArrayOutputStream baos = new ByteArrayOutputStream();
115     ObjectOutputStream oos = new ObjectOutputStream(baos);
116
117     oos.writeObject(key_pair);
118     oos.writeObject(cert);
119     System.out.println(cert.getClass().getName());
120     oos.close();
121
122     byte[] iv_bytes = "this is the iv".getBytes();
123     SecretKeySpec sks = new SecretKeySpec(passphrase.getBytes(), 0, 8, "DES");
124     IvParameterSpec ap = new IvParameterSpec(iv_bytes, 0, 8);
125
126     Cipher c = Cipher.getInstance("DES/CBC/PKCS5Padding");
127     SecureRandom sr = new SecureRandom("This is a very bad seed".getBytes());
128     c.init(Cipher.ENCRYPT_MODE, sks, ap, sr);
129
130     byte[] final_bytes = c.doFinal(baos.toByteArray());
131
132     FileOutputStream fos = new FileOutputStream(filename + ".info");
133     fos.write(final_bytes);
134     fos.close();
135
136     fos = new FileOutputStream(filename + ".crt");
137     fos.write(cert.toByteArray());
138     fos.close();
139
140     File f = new File(filename + ".crl");
141     f.createNewFile();
142
143   }
144   catch(Exception e)
145   {
146     e.printStackTrace();
147   }
```

```
148  }
149
150  public static void printCAInfo(String passphrase, String filename)
151  {
152    try
153    {
154      FileInputStream fis = new FileInputStream(filename + ".info");
155
156      int b = fis.read();
157
158      ByteArrayOutputStream baos = new ByteArrayOutputStream();
159
160      while( b != -1)
161      {
162          baos.write(b);
163          b= fis.read();
164      }
165      //DECRYPT
166      byte[] iv_bytes = "this is the iv".getBytes();
167      SecretKeySpec sks = new SecretKeySpec(passphrase.getBytes(), 0, 8, "DES");
168      IvParameterSpec ap = new IvParameterSpec(iv_bytes, 0, 8 );
169
170      Cipher c = Cipher.getInstance("DES/CBC/PKCS5Padding");
171      SecureRandom sr = new SecureRandom("This is a very bad seed".getBytes());
172      c.init(Cipher.DECRYPT_MODE, sks, ap, sr);
173
174      byte[] final_bytes = c.doFinal(baos.toByteArray());
175
176      ByteArrayInputStream bais = new ByteArrayInputStream(final_bytes);
177      ObjectInputStream ois = new ObjectInputStream(bais);
178
179      KeyPair kp = (KeyPair)ois.readObject();
180      Object cert = ois.readObject();
181      System.out.println(cert.getClass().getName());
182
183      System.out.println("--------------------");
184      System.out.println(cert);
185      System.out.println("--------------------");
186
187    }
188    catch(Exception e)
189    {
190      e.printStackTrace();
191    }
192  }
193
194  public CertIssuer createIssuer()
195  {
196    return new CertIssuer(m_keys, m_cert);
197  }
```

```
198
199   public static void main(String args[])
200   {
201     CA generateAndStore(args[0], args[1]),
202     CA printCAInfo(args[0], args[1]),
203   }
204 }
```

C 3 2 CertIssuer java

```
1 package pki,
2
3 import pki *,
4
5 import java security *,
6 import javax crypto spec *,
7 import javax crypto *,
8 import java io *,
9 import iaik pkcs pkcs10 CertificateRequest,
10 import iaik asn1 structures *,
11 import iaik asn1 *,
12 import java security cert *,
13
14 import java util *,
15 import java math *,
16
17 public class CertIssuer implements Serializable
18 {
19   private KeyPair m_keys,
20   private X509Certificate m_cert,
21
22   public CertIssuer(KeyPair keys, X509Certificate cert)
23   {
24     m_keys = keys,
25     m_cert = cert,
26   }
27
28   public PublicKey getPublic()
29   {
30     return m_keys getPublic(),
31   }
32
33   public X509Certificate getCert()
34   {
35     return m_cert,
36   }
37
38   public byte[] processCertificateRequest(byte[] request)
39   {
40     try
41     {
42       CertificateRequest cert_request = new CertificateRequest(request),
43
44       Name subject = cert_request getSubject(),
45       PublicKey pk = cert_request getPublicKey(),
46
47       String l = new String(),
48       l += System currentTimeMillis(),
49
```

```
50      SecureRandom sr = new SecureRandom(l getBytes()),
51      String serial = new String(),
52      serial += sr nextLong(),
53
54      iaik x509 X509Certificate t_cert =
55        new iaik x509 X509Certificate(m_cert getEncoded()),
56      iaik asn1 structures Name issuer =
57        (iaik asn1 structures Name)t_cert getSubjectDN(),
58
59      iaik x509 X509Certificate cert = new iaik x509 X509Certificate(),
60      cert setIssuerDN(issuer),
61      cert setSubjectDN(subject),
62      cert setPublicKey(pk),
63      cert setSerialNumber(new BigInteger(serial)),
64      GregorianCalendar date = (GregorianCalendar)Calendar getInstance(),
65      date add(Calendar MONTH, -1),
66      cert setValidNotBefore(date getTime()),
67      date add(Calendar MONTH, 5),
68      cert setValidNotAfter(date getTime()),
69      cert sign(AlgorithmID sha1WithRSAEncryption, m_keys getPrivate()),
70
71      return cert toByteArray(),
72    }
73    catch(Exception e)
74    {
75      e printStackTrace(),
76    }
77    return null,
78  }
79}
```

C 3 3 Client java

```
1 package pki;
2
3 import pki *;
4
5 import java security *;
6 import java security interfaces *;
7 import javax crypto spec *;
8 import javax crypto *;
9 import java io *;
10 import iaik pkcs pkcs10 CertificateRequest;
11 import iaik asn1 structures *;
12 import iaik asn1 *;
13 import java security cert *;
14
15 public class Client implements Serializable, PkiBase
16 {
17   private KeyPair m_keys = null;
18   private X509Certificate m_cert = null;
19   private X509Certificate ca_cert = null;
20
21   public Client(){}
22
23   public Client(String passphrase, String filename, String cafn)
24   {
25     try{
26       FileInputStream fis = new FileInputStream(cafn + " crt");
27       ByteArrayOutputStream bt = new ByteArrayOutputStream();
28
29       int b = fis read();
30
31       while( b != -1)
32       {
33         bt write(b);
34         b = fis read();
35       }
36
37       fis close();
38       bt close();
39       \
40       byte[] cacert_bytes = bt toByteArray();
41
42       fis = new FileInputStream(filename + " crt");
43       bt = new ByteArrayOutputStream();
44
45       b = fis read();
46
47       while( b != -1)
48       {
49         bt write(b);
```

```
50          b = fis read(),
51        }
52
53        fis close(),
54        bt close(),
55
56        byte[] cert_bytes = bt toByteArray(),
57
58        fis = new FileInputStream(filename + " key"),
59        bt = new ByteArrayOutputStream(),
60
61        b = fis read(),
62
63        while( b != -1)
64        {
65          bt write(b),
66          b = fis read(),
67        }
68
69        byte[] iv_bytes = "this is the iv" getBytes(),
70        SecretKeySpec sks = new SecretKeySpec(passphrase getBytes(), 0, 8, "DES"),
71        IvParameterSpec ap = new IvParameterSpec(iv_bytes, 0, 8 ),
72
73        Cipher c = Cipher getInstance("DES/CBC/PKCS5Padding"),
74        SecureRandom sr = new SecureRandom("This is a very bad seed" getBytes()),
75        c init(Cipher DECRYPT_MODE, sks, ap, sr),
76
77        byte[] final_bytes = c doFinal(bt toByteArray()),
78
79        ByteArrayInputStream bais = new ByteArrayInputStream(final_bytes),
80        ObjectInputStream ois = new ObjectInputStream(bais),
81
82        KeyPair kp = (KeyPair)ois readObject(),
83        CertificateFactory cf = CertificateFactory getInstance("X509"),
84        ByteArrayInputStream cbais = new ByteArrayInputStream(cert_bytes),
85        X509Certificate cert = (X509Certificate)cf generateCertificate(cbais),
86
87        cbais = new ByteArrayInputStream(cacert_bytes),
88        X509Certificate cacert = (X509Certificate)cf generateCertificate(cbais),
89
90        cert verify(cacert getPublicKey()),
91        m_cert = cert,
92        m_keys = kp,
93        ca_cert = cacert,
94      }
95      catch(Exception e){e printStackTrace(),}
96    }
97
98    public Client(String passphrase, String filename, byte[] cacert_bytes)
99    {
```

```
100     try
101     {
102       FileInputStream fis = new FileInputStream(filename + " crt"),
103       ByteArrayOutputStream bt = new ByteArrayOutputStream(),
104
105       int b = fis read(),
106
107       while( b != -1)
108       {
109         bt write(b),
110         b = fis read(),
111       }
112
113       fis close(),
114       bt close(),
115
116       byte[] cert_bytes = bt toByteArray(),
117
118       fis = new FileInputStream(filename + " key"),
119       bt = new ByteArrayOutputStream(),
120
121       b = fis read(),
122
123       while( b != -1)
124       {
125         bt write(b),
126         b = fis read(),
127       }
128
129       byte[] iv_bytes = "this is the iv" getBytes(),
130       SecretKeySpec sks = new SecretKeySpec(passphrase getBytes(), 0, 8, "DES"),
131       IvParameterSpec ap = new IvParameterSpec(iv_bytes, 0, 8 ),
132
133       Cipher c = Cipher getInstance("DES/CBC/PKCS5Padding"),
134       SecureRandom sr = new SecureRandom("This is a very bad seed" getBytes()),
135       c init(Cipher DECRYPT_MODE, sks, ap, sr),
136
137       byte[] final_bytes = c doFinal(bt toByteArray()),
138
139       ByteArrayInputStream bais = new ByteArrayInputStream(final_bytes),
140       ObjectInputStream ois = new ObjectInputStream(bais),
141
142       KeyPair kp = (KeyPair)ois readObject(),
143       CertificateFactory cf = CertificateFactory getInstance("X509"),
144       ByteArrayInputStream cbais = new ByteArrayInputStream(cert_bytes),
145       X509Certificate cert = (X509Certificate)cf generateCertificate(cbais),
146
147       cbais = new ByteArrayInputStream(cacert_bytes),
148       X509Certificate cacert = (X509Certificate)cf generateCertificate(cbais),
149
```

```
150       cert verify(cacert getPublicKey()),
151       m_cert = cert,
152       m_keys = kp,
153       ca_cert = cacert,
154
155     }
156     catch(Exception e)
157     {
158       e printStackTrace(),
159     }
160   }
161
162   public Client(byte[] certBytes, byte[] keypair, String passphrase)
163   {
164     try
165     {
166       byte[] iv_bytes = "this is the iv" getBytes(),
167       SecretKeySpec sks = new SecretKeySpec(passphrase getBytes(), 0, 8, "DES"),
168       IvParameterSpec ap = new IvParameterSpec(iv_bytes, 0, 8 ),
169
170       Cipher c = Cipher getInstance("DES/CBC/PKCS5Padding"),
171       SecureRandom sr = new SecureRandom("This is a very bad seed" getBytes()),
172       c init(Cipher DECRYPT_MODE, sks, ap, sr),
173
174       byte[] final_bytes = c doFinal(keypair),
175
176       ByteArrayInputStream bais = new ByteArrayInputStream(final_bytes),
177       ObjectInputStream ois = new ObjectInputStream(bais),
178
179       KeyPair kp = (KeyPair)ois readObject(),
180       CertificateFactory cf = CertificateFactory getInstance("X509"),
181       ByteArrayInputStream cbais = new ByteArrayInputStream(certBytes),
182       X509Certificate cert = (X509Certificate)cf generateCertificate(cbais),
183
184       m_cert = cert,
185       m_keys = kp,
186     }
187     catch(Exception e)
188     {
189       e printStackTrace(),
190     }
191   }
192
193   public KeyPair generateKeyPair(int len)
194   {
195     try
196     {
197       String seed = new String(),
198       seed += System currentTimeMillis(),
199       SecureRandom sec_random = new SecureRandom(seed getBytes()),
```

```
200
201       KeyPairGenerator key_gen = KeyPairGenerator getInstance("RSA");
202       key_gen initialize(len, sec_random);
203       m_keys = key_gen generateKeyPair();
204     }
205     catch(Exception e)
206     {
207       e printStackTrace();
208     }
209     return m_keys;
210   }
211
212   public byte[] generateCertificateRequest(byte[] name)
213   {
214     try
215     {
216       if(m_keys == null)
217       {
218         generateKeyPair(1028);
219       }
220       Name n = new Name(name);
221       CertificateRequest c = new CertificateRequest(m_keys getPublic(),n);
222       c sign(AlgorithmID sha1WithRSAEncryption,m_keys getPrivate());
223
224       byte[] bytes = c toByteArray();
225
226       return bytes;
227     }
228     catch(Exception e)
229     {
230       e printStackTrace();
231       return null;
232     }
233   }
234
235   public void setCertificate(byte[] c, byte[] ca)
236   {
237     try
238     {
239       CertificateFactory cf = CertificateFactory getInstance("X509");
240       ByteArrayInputStream bais = new ByteArrayInputStream(c);
241       X509Certificate cert = (X509Certificate)cf generateCertificate(bais);
242
243       bais = new ByteArrayInputStream(ca);
244       X509Certificate cacert = (X509Certificate)cf generateCertificate(bais);
245
246       setCertificate(cert,cacert);
247     }
248     catch(Exception e)
249     {
```

```
250       e printStackTrace(),
251     }
252   }
253
254   public static void verifyCertIssuer(byte[] cabytes, byte[] bytes) throws Exception
255   {
256     CertificateFactory cf = CertificateFactory getInstance("X509"),
257     ByteArrayInputStream bais = new ByteArrayInputStream(cabytes),
258     X509Certificate cacert = (X509Certificate)cf generateCertificate(bais),
259
260     bais = new ByteArrayInputStream(bytes),
261     X509Certificate cert = (X509Certificate)cf generateCertificate(bais),
262
263     cert verify(cacert getPublicKey()),
264   }
265
266   public void setCertificate(X509Certificate c, X509Certificate ca) throws Exception
267   {
268     c verify(ca getPublicKey()),
269
270     if( c getPublicKey() equals(m_keys getPublic()))
271     {
272       m_cert = c,
273     }
274     else
275     {
276       throw new Exception("Invalid cert"),
277     }
278   }
279
280   public void store(String passphrase, String prefix)
281   {
282     try
283     {
284
285       if(m_keys != null)
286       {
287         ByteArrayOutputStream baos = new ByteArrayOutputStream(),
288         ObjectOutputStream oos = new ObjectOutputStream(baos),
289
290         oos writeObject(m_keys),
291         oos close(),
292
293         SecretKeySpec new_sks = new SecretKeySpec( passphrase getBytes(), 0, 8, "DES"),
294         SecureRandom sr = new SecureRandom("this is a very bad seed" getBytes()),
295         byte[] iv_bytes = "this is the iv" getBytes(),
296
297         IvParameterSpec ap = new IvParameterSpec(iv_bytes,0,8),
298         Cipher c  = Cipher getInstance("DES/CBC/PKCS5Padding"),
299         c init(Cipher ENCRYPT_MODE, new_sks, ap, sr),
```

```
300         byte[] bytes = c.doFinal(baos.toByteArray());
301
302
303         FileOutputStream fos = new FileOutputStream( new String( prefix + " key"));
304         fos.write(bytes);
305         fos.close();
306       }
307
308       if( m_cert != null)
309       {
310         FileOutputStream fos = new FileOutputStream( new String( prefix + " crt"));
311         fos.write(m_cert.getEncoded());
312         fos.close();
313       }
314     }
315     catch(Exception e)
316     {
317       e.printStackTrace();
318     }
319   }
320
321   public byte[] getCertBytes()
322   {
323     try
324     {
325       return getCert().getEncoded();
326     }
327     catch(Exception e)
328     {
329       e.printStackTrace();
330       return null;
331     }
332   }
333
334   public X509Certificate getCert()
335   {
336     return m_cert;
337   }
338
339   public KeyPair getKeys()
340   {
341     return m_keys;
342   }
343
344   public static byte[] loadCACert(String filename)
345   {
346     try
347     {
348       FileInputStream fis = new FileInputStream(filename + " crt");
349       ByteArrayOutputStream bt = new ByteArrayOutputStream();
```

```
350
351       int b = fis read();
352
353       while( b != -1)
354       {
355         bt write(b);
356         b = fis read();
357       }
358
359       fis close();
360       bt close();
361
362       byte[] cert_bytes = bt toByteArray();
363
364       return cert_bytes;
365     }
366     catch(Exception e)
367     {
368       e printStackTrace();
369       return null;
370     }
371   }
372
373   public static String getCAFileName()
374   {
375     try
376     {
377       LineNumberReader lnr =
378         new LineNumberReader(new InputStreamReader(System in));
379       System out println("Enter the location of the CA certificate ");
380       String cacertfilename = lnr readLine();
381
382       return cacertfilename;
383     }
384     catch(Exception e)
385     {
386       e printStackTrace();
387       return null;
388     }
389   }
390
391   public static byte[] getName()
392   {
393     try
394     {
395       LineNumberReader lnr =
396         new LineNumberReader(new InputStreamReader(System in));
397
398       System out println("Enter country code  e g  IE");
399       String cc = lnr readLine();
```

```
400       System out println("Enter locality  e g  Dublin"),
401       String loc = lnr readLine(),
402       System out println("Enter organization  e g  DCU"),
403       String org = lnr readLine(),
404       System out println("Enter organizational unit  e g  POSTGRAD"),
405       String unit = lnr readLine(),
406       System out println("Enter common name  e g  John Doe"),
407       String cn = lnr readLine(),
408
409       Name name = new Name(),
410       name addRDN(ObjectID country, cc),
411       name addRDN(ObjectID locality, loc),
412       name addRDN(ObjectID organization ,org),
413       name addRDN(ObjectID organizationalUnit ,unit),
414       name addRDN(ObjectID commonName ,cn),
415
416       return name getEncoded(),
417     }
418     catch(Exception e)
419     {
420      e printStackTrace(),
421     }
422     return null,
423   }
424
425   public static String getPassPhrase()
426   {
427    try
428    {
429       LineNumberReader lnr = new LineNumberReader(new InputStreamReader(System in)),
430
431       System out println("Enter your passphrase "),
432       String pp = lnr readLine(),
433
434       return pp,
435     }
436     catch(Exception e)
437     {
438      return null,
439     }
440   }
441
442   public static String getFilename()
443   {
444     try
445     {
446       LineNumberReader lnr = new LineNumberReader(new InputStreamReader(System in)),
447
448       System out println("Enter the prefix for all client files "),
449       String pp = lnr readLine(),
```

```
450
451       return pp;
452     }
453     catch(Exception e)
454     {
455       return null;
456     }
457   }
458
459   public byte[] createRequest(byte[] serviceCertBytes, Long l)
460   {
461     try
462     {
463       CertificateFactory cf = CertificateFactory getInstance("X509");
464       ByteArrayInputStream bais = new ByteArrayInputStream(serviceCertBytes);
465       X509Certificate cert = (X509Certificate)cf generateCertificate(bais);
466
467       ByteArrayOutputStream baos = new ByteArrayOutputStream();
468       DataOutputStream dos = new DataOutputStream(baos);
469
470       dos writeInt(0);
471       dos writeLong(l longValue());
472
473       Principal n = (Principal)cert getSubjectDN();
474       System out println(n getName());
475
476       dos writeUTF(n getName());
477
478       byte[] data = (new String("data")) getBytes();
479
480       byte[] encData = Enc encryptData(data,cert getPublicKey());
481
482       dos writeInt(encData length);
483       dos write(encData, 0, encData length);
484
485       dos close();
486
487
488       byte[] to_enc = baos toByteArray();
489
490       byte[] to_return = Enc encryptData(to_enc,m_keys getPrivate());
491
492       return to_return;
493     }
494     catch(Exception e)
495     {
496       e printStackTrace();
497     }
498     return null;
499   }
```

```
500
501 public byte[] createClientResponse(byte[] serviceCertBytes, byte[] packet, Long randA)
502 {
503   try
504   {
505     CertificateFactory cf = CertificateFactory getInstance("X509"),
506     ByteArrayInputStream bais = new ByteArrayInputStream(serviceCertBytes),
507     X509Certificate cert = (X509Certificate)cf generateCertificate(bais),
508
509     byte[] decData = Enc decryptData(packet,cert getPublicKey()),
510     bais = new ByteArrayInputStream(decData),
511     DataInputStream dis = new DataInputStream(bais),
512
513     Long randB = new Long(dis readLong()),
514     System out println(randB),
515
516     String ib= dis readUTF(),
517     String ia= dis readUTF(),
518
519     Long randATest = new Long(dis readLong()),
520
521     int lenEnc = dis readInt(),
522     byte[] encData = new byte[lenEnc],
523     dis read(encData,0,lenEnc),
524
525     byte[] dummyData = Enc decryptData(encData,m_keys getPrivate()),
526     sun security x509 X500Name testIa
527       = (sun security x509 X500Name)m_cert getSubjectDN(),
528     sun security x509 X500Name testIb
529       = (sun security x509 X500Name)cert getSubjectDN(),
530
531     if('ib equals(testIb getName()))
532       throw new Exception("NAMES NOT EQUAL"),
533
534     if('ia equals(testIa getName()))
535       throw new Exception("NAMES NOT EQUAL"),
536
537     if('randATest equals(randA))
538       throw new Exception("RANDOM CHALLENGE FAILED"),
539
540     ByteArrayOutputStream baos = new ByteArrayOutputStream(),
541     DataOutputStream dos = new DataOutputStream(baos),
542
543     dos writeLong(randB longValue()),
544     dos close(),
545
546     byte[] to_return = Enc encryptData(baos toByteArray(),m_keys getPrivate()),
547
548     return to_return,
549   }
```

```
550   catch(Exception e)
551   {
552    e printStackTrace(),
553   }
554
555   return null,
556  }
557
558  public byte[] extractKey(byte[] serviceCertBytes, byte[] encKey)
559  {
560    try
561    {
562      CertificateFactory cf = CertificateFactory getInstance("X509"),
563      ByteArrayInputStream bais = new ByteArrayInputStream(serviceCertBytes),
564      X509Certificate cert = (X509Certificate)cf generateCertificate(bais),
565
566      byte[] decData = Enc decryptData(encKey,cert getPublicKey()),
567
568      byte[] to_return = Enc decryptData(decData,m_keys getPrivate()),
569                         byte[] to_return = c doFinal(decData),
570      return to_return,
571    }
572    catch(Exception e)
573    {
574      e printStackTrace(),
575    }
576    return null,
577  }
578
579  public static boolean chooseFirst()
580  {
581    try
582    {
583      System out println("Use service one (Y/N) "),
584
585      LineNumberReader lnr = new LineNumberReader(new InputStreamReader(System in)),
586
587      String resp = lnr readLine(),
588
589      while('((resp equalsIgnoreCase("Y"))||(resp equalsIgnoreCase("N"))))
590      {
591        System out println("Use service one (Y/N) "),
592        resp = lnr readLine(),
593      }
594
595    if(resp equalsIgnoreCase("Y"))
596      return true,
597    else
598      return false,
599   }
```

```
600     catch(Exception e)
601     {
602      e printStackTrace(),
603     }
604
605     return true,
606  }
607 }
```

C 3 4 SP.java

```
1 package pki;
2
3 import pki.*;
4
5 import java.security.*;
6 import javax.crypto.spec.*;
7 import javax.crypto.*;
8 import java.io.*;
9 import iaik.pkcs.pkcs10.CertificateRequest;
10 import iaik.asn1.structures.*;
11 import iaik.asn1.*;
12 import java.security.cert.*;
13
14 public class SP implements Serializable, PkiBase
15 {
16   private KeyPair m_keys = null;
17   private X509Certificate m_cert = null;
18   private X509Certificate ca_cert = null;
19
20   public SP(byte[] ca_cert_bytes)
21   {
22     try
23     {
24       CertificateFactory cf = CertificateFactory.getInstance("X509");
25       ByteArrayInputStream bais = new ByteArrayInputStream(ca_cert_bytes);
26       ca_cert = (X509Certificate)cf.generateCertificate(bais);
27     }
28     catch(Exception e)
29     {
30       e.printStackTrace();
31     }
32   }
33
34   public KeyPair generateKeyPair(int len)
35   {
36     try
37     {
38       String seed = new String();
39       seed += System.currentTimeMillis();
40       SecureRandom sec_random = new SecureRandom(seed.getBytes());
41
42       KeyPairGenerator key_gen = KeyPairGenerator.getInstance("RSA");
43       key_gen.initialize(len, sec_random);
44       m_keys = key_gen.generateKeyPair();
45     }
46     catch(Exception e)
47     {
48       e.printStackTrace();
49     }
```

```
50    return m_keys;
51  }
52
53  public byte[] generateCertificateRequest()
54  {
55    try
56    {
57      if(m_keys == null)
58      {
59        generateKeyPair(1028);
60      }
61      Name n = new Name(this.getName());
62      CertificateRequest c = new CertificateRequest(m_keys.getPublic(),n);
63      c.sign(AlgorithmID.sha1WithRSAEncryption,m_keys.getPrivate());
64
65      byte[] bytes = c.toByteArray();
66
67      return bytes;
68    }
69    catch(Exception e)
70    {
71      e.printStackTrace();
72      return null;
73    }
74  }
75
76  public void setCertificate(byte[] c, byte[] ca)
77  {
78    try
79    {
80      CertificateFactory cf = CertificateFactory.getInstance("X509");
81      ByteArrayInputStream bais = new ByteArrayInputStream(c);
82      X509Certificate cert = (X509Certificate)cf.generateCertificate(bais);
83
84      bais = new ByteArrayInputStream(ca);
85      X509Certificate cacert = (X509Certificate)cf.generateCertificate(bais);
86
87      setCertificate(cert,cacert);
88    }
89    catch(Exception e)
90    {
91      e.printStackTrace();
92    }
93  }
94
95  public void setCertificate(X509Certificate c, X509Certificate ca) throws Exception
96  {
97    c.verify(ca.getPublicKey());
98
99    if( c.getPublicKey().equals(m_keys.getPublic()))
```

```
100     {
101       m_cert = c;
102     }
103     else
104     {
105       throw new Exception("Invalid cert");
106     }
107   }
108
109   public byte[] getCertBytes()
110   {
111     try
112     {
113       return getCert().getEncoded();
114     }
115     catch(Exception e)
116     {
117       e.printStackTrace();
118       return null;
119     }
120   }
121
122   public X509Certificate getCert()
123   {
124     return m_cert;
125   }
126
127   public KeyPair getKeys()
128   {
129     return m_keys;
130   }
131
132   public static byte[] loadCACert(String filename)
133   {
134     try
135     {
136       FileInputStream fis = new FileInputStream(filename + ".crt");
137       ByteArrayOutputStream bt = new ByteArrayOutputStream();
138
139       int b = fis.read();
140
141       while( b != -1)
142       {
143         bt.write(b);
144         b = fis.read();
145       }
146
147       fis.close();
148       bt.close();
149
```

```
150       byte[] cert_bytes = bt toByteArray();
151
152       return cert_bytes;
153     }
154     catch(Exception e)
155     {
156       e printStackTrace();
157       return null;
158     }
159   }
160
161   public static String getCAFileName()
162   {
163     try
164     {
165       LineNumberReader lnr = new LineNumberReader(new InputStreamReader(System in));
166       System out println("Enter the location of the CA certificate ");
167       String cacertfilename = lnr readLine();
168
169       return cacertfilename;
170     }
171     catch(Exception e)
172     {
173       e printStackTrace();
174       return null;
175     }
176   }
177
178   public static byte[] getName()
179   {
180     try
181     {
182       LineNumberReader lnr =
183         new LineNumberReader(new InputStreamReader(System in));
184
185       String cc = "ie";
186       String loc = "Dublin";
187       String org = "DCU";
188       String unit = "PostGrad";
189       String cn = "Example Service Provider";
190
191       Name name = new Name();
192       name addRDN(ObjectID country, cc);
193       name addRDN(ObjectID locality, loc);
194       name addRDN(ObjectID organization ,org);
195       name addRDN(ObjectID organizationalUnit ,unit);
196       name addRDN(ObjectID commonName ,cn);
197
198       return name getEncoded();
199     }
```

```
200     catch(Exception e)
201     {
202       e printStackTrace(),
203     }
204     return null,
205   }
206
207   public static String getPassPhrase()
208   {
209     try
210     {
211       LineNumberReader lnr =
212         new LineNumberReader(new InputStreamReader(System in)),
213
214       System out println("Enter your passphrase "),
215       String pp = lnr readLine(),
216
217       return pp,
218     }
219     catch(Exception e)
220     {
221       return null,
222     }
223   }
224
225   public static String getFilename()
226   {
227     try
228     {
229       LineNumberReader lnr =
230        new LineNumberReader(new InputStreamReader(System in)),
231
232       System out println("Enter the prefix for all client files "),
233       String pp = lnr readLine(),
234
235       return pp,
236     }
237     catch(Exception e)
238     {
239       return null,
240     }
241   }
242
243   public byte[] encryptKeyBytes(byte[] clientCertBytes, byte[] keyBytes)
244   {
245     try
246     {
247       CertificateFactory cf = CertificateFactory getInstance("X509"),
248       ByteArrayInputStream bais = new ByteArrayInputStream(clientCertBytes),
249       X509Certificate cert = (X509Certificate)cf generateCertificate(bais),
```

```
250
251       byte[] encData = Enc encryptData(keyBytes,cert getPublicKey()),
252       byte[] to_return = Enc encryptData(encData,m_keys getPrivate()),
253       return to_return,
254     }
255     catch(Exception e)
256     {
257       e printStackTrace(),
258     }
259
260     return null,
261   }
262
263   public byte[] processClientResponse(byte[] clientCertBytes, byte[] packet, Long randB)
264   {
265     try
266     {
267       CertificateFactory cf = CertificateFactory getInstance("X509"),
268       ByteArrayInputStream bais = new ByteArrayInputStream(clientCertBytes),
269       X509Certificate cert = (X509Certificate)cf generateCertificate(bais),
270
271       byte[] decData = Enc decryptData(packet,cert getPublicKey()),
272
273       bais = new ByteArrayInputStream(decData),
274       DataInputStream dis = new DataInputStream(bais),
275
276       Long testRandB = new Long(dis readLong()),
277
278       if(!testRandB equals(randB))
279       throw new Exception("RANDOMS NOT THE SAME"),
280
281       SecureRandom random = SecureRandom getInstance("SHA1PRNG"),
282       byte[] key = new byte[16],
283       random setSeed(System currentTimeMillis()),
284
285       random nextBytes(key),
286
287       return key,
288     }
289     catch(Exception e)
290     {
291       e printStackTrace(),
292     }
293
294     return null,
295   }
296
297   public byte[] createServiceResponse(Long randA, Long randB, byte[] clientCertBytes)
298   {
299     try
```

```
300     {
301       CertificateFactory cf = CertificateFactory getInstance("X509"),
302       ByteArrayInputStream bais = new ByteArrayInputStream(clientCertBytes),
303       X509Certificate cert = (X509Certificate)cf generateCertificate(bais),
304
305       byte[] encData = Enc encryptData(randB toString() getBytes(),cert getPublicKey()),
306       ByteArrayOutputStream baos = new ByteArrayOutputStream(),
307       DataOutputStream dos = new DataOutputStream(baos),
308
309       sun security x509 X500Name ib = (sun security x509 X500Name)m_cert getSubjectDN(),
310       sun security x509 X500Name ia = (sun security x509 X500Name)cert getSubjectDN(),
311
312       dos writeLong(randB longValue()),
313       System out println(randB),
314
315       dos writeUTF(ib getName()),
316
317       dos writeUTF(ia getName()),
318
319       dos writeLong(randA longValue()),
320
321       dos writeInt(encData length),
322       dos write(encData,0,encData length),
323
324       dos close(),
325
326       byte[] to_return = Enc encryptData(baos toByteArray(),m_keys getPrivate()),
327       return to_return,
328
329     }
330     catch(Exception e)
331     {
332       e printStackTrace(),
333       return null,
334     }
335   }
336
337   public Long processClientRequest(byte[] request, byte[] clientCertBytes)
338   {
339     try
340     {
341       CertificateFactory cf = CertificateFactory getInstance("X509"),
342       ByteArrayInputStream bais = new ByteArrayInputStream(clientCertBytes),
343       X509Certificate cert = (X509Certificate)cf generateCertificate(bais),
344
345       byte[] decData = Enc decryptData(request,cert getPublicKey()),
346       ByteArrayInputStream encIn = new ByteArrayInputStream(request),
347       DataInputStream encDis = new DataInputStream(encIn),
348
349       bais = new ByteArrayInputStream(decData),
```

```
350       DataInputStream dis = new DataInputStream(bais),
351
352       int ta = dis readInt(),
353
354       long l = dis readLong(),
355
356       String ib= dis readUTF(),
357
358       sun security x509 X500Name toTest
359         = (sun security x509 X500Name)m_cert getSubjectDN(),
360
361       if('ib equals(toTest getName()))
362         throw new Exception("NOT MEANT FOR ME"),
363
364       int len2 = dis readInt(),
365
366       byte[] encData = new byte[len2],
367
368       dis read(encData,0,len2),
369
370       byte[] dummyData = Enc decryptData(encData,m_keys getPrivate()),
371       return new Long(l),
372     }
373     catch(Exception e)
374     {
375       e printStackTrace(),
376     }
377     return null,
378   }
379 }
```

C 3 5 PkiBase java

```
1 package pki;
2
3 import java.security.*;
4 import javax.crypto.spec.*;
5 import javax.crypto.*;
6 import java.io.*;
7 import iaik.pkcs.pkcs10.CertificateRequest;
8 import iaik.asn1.structures.*;
9 import iaik.asn1.*;
10 import java.security.cert.*;
11
12 public interface PkiBase extends Serializable
13 {
14   public void setCertificate(byte[] c, byte[] ca);
15
16   public byte[] getCertBytes();
17 }
```

C 3 6 Enc java

```
1 package pki;
2
3 import java security *;
4 import java security interfaces *;
5 import javax crypto spec *;
6 import javax crypto *;
7 import java io *;
8 import iaik pkcs pkcs10 CertificateRequest;
9 import iaik asn1 structures *;
10 import iaik asn1 *;
11 import java security cert *;
12
13 public class Enc
14 {
15   public static byte[] decryptData(byte[] data, Key k) throws Exception
16   {
17     int blocksize = 32;
18
19     ByteArrayInputStream bais = new ByteArrayInputStream(data);
20     ObjectInputStream ois = new ObjectInputStream(bais);
21
22     Object o = ois readObject();
23     Cipher c = Cipher getInstance("RSA");
24     c init(Cipher DECRYPT_MODE,k);
25
26     ByteArrayOutputStream baos = new ByteArrayOutputStream();
27
28     while(o != null)
29     {
30       byte[] bytes = (byte[])o;
31
32       byte[] decData = c doFinal(bytes);
33
34       baos write(decData,0,decData length);
35
36       try
37       {
38         o = ois readObject();
39       }
40       catch(Exception e)
41       {
42         o = null;
43       }
44     }
45     return baos toByteArray();
46   }
47
48   public static byte[] encryptData(byte[] data, Key k) throws Exception
49   {
```

```
50      int blocksize = 32,
51
52      ByteArrayOutputStream baos = new ByteArrayOutputStream(),
53      ObjectOutputStream oos = new ObjectOutputStream(baos),
54
55      Cipher c = Cipher getInstance("RSA"),
56
57      c init(Cipher ENCRYPT_MODE,k),
58
59      int count = 0,
60
61      for(count = 0, count < (data length - blocksize), count += blocksize)
62      {
63        byte[] encData = c doFinal(data,count,blocksize),
64        oos writeObject(encData),
65      }
66      byte[] final_data = c doFinal(data,count,(data length - count)),
67      oos writeObject(final_data),
68      return baos toByteArray(),
69    }
70}
```

Bibliography

Abadi, M & Gordon, A D (1997), A calculus for cryptographic protocols The spi calculus, *in* 'Fourth ACM Conference on Computer and Communications Security', ACM Press, pp 36–47

Abadi, M & Gordon, A D (1998), A calculus for cryptographic protocols The spi calculus, Technical Report 149, digital Systems Research Centre

Black, P E , Hall, K M , Jones, M D & Windley, T N L P J (1996), A brief introduction to formal methods, *in* 'Proceedings of the IEEE 1996 Custom Integrated Circuits Conference', p 377

Butler, R W , Caldwell, J L , Carreno, V A , Holloway, C M & Miner, P S (1995), NASA langley's research and technology-transfer program in formal methods, *in* 'Proceedings of Tenth Annual Conference on Computer Assurance (COMPASS 95)', NASA Langley Research Center

Butler, R W , Caldwell, J L , Carreno, V A , Holloway, C M & Miner, P S (1998), 'Nasa langley's research and technology-transfer program in formal methods (updated version of 1995 paper of same name)'

Cardelli, L & Gordon, A D (1998), Mobile ambients, *in* 'Foundations of Software Science and Computation Structures First International Conference, FOSSACS '98', Springer-Verlag, Berlin Germany
*citeseer ist psu edu/cardelli98mobile html

Fournet, C & Gonthier, G (1996), The reflexive CHAM and the join-calculus, *in* 'Proceedings of the 23rd ACM Symposium on Principles of Programming Languages', ACM Press, pp 372–385
*citeseer ist psu edu/fournet95reflexive html

Hall, A (1990), 'Seven myths of formal systems', *IEEE Software* p 11

Heitmeyer, C (1998), On the need for practical formal methods, *in* 'Proceedings of the 5th International Formal Techniques in Real-time and Real-time fault-tolerant systems Symposium', p 18

Hoare, C (1985), *Communicating Sequential Processes*, Prentice Hall

JPL, N (n d), 'Formal methods specification and verification guidebook for software and computer systems, volume i Planning and technology insertion', Available from http //eis jpl nasa gov/quality/Formal_Methods/ Accessed 28th April 2003

Milner, R (1989), *Communication and Concurrency*, Prentice Hall

Milner, R (1993), The polyadic pi-calculus A tutorial, *in* F L Bauer, W Brauer & H Schwichtenberg, eds, 'Logic and Algebra of Specification', Springer, Berlin Heidelberg, pp 203–246

Milner, R (1999), *Communicating and Mobile Systesm The Pi-calculus*, Cambridge University Press

Milner, R , Parrow, J & Walker, D (1989), A calculus of mobile processes parts i and ii, Technical Report -86, University of Edinburgh

Nierstrasz, O , Achermann, F & Kneubuehl, S (n d), 'A guide to jpiccola' *citeseer ist psu edu/nierstrasz03guide html

Parrow, J (2001), *Handbook of Process Algebra*, Bergstra Ponse Smolka, chapter An Introduction to the pi-Calculus, p 479

Parrow, J & Victor, B (1998), The fusion calculus Expressiveness and symmetry in mobile processes, *in* 'Logic in Computer Science', pp 176–185 *citeseer ist psu edu/parrow97fusion html

Paulson, L C (1996), *ML for the Working Programmer* , Cambridge University Press

Pierce, B C (1997), Programming in the pi-calculus A tutorial introduction to Pict

Pierce, B C & Turner, D N (2000*a*), Pict A programming language based on the pi-calculus, *in* G Plotkin, C Stirling & M Tofte, eds, 'Proof, Language and Interaction Essays in Honour of Robin Milner', MIT Press *citeseer ist psu edu/pierce97pict html

Pierce, B C & Turner, D N (2000*b*), Pict A programming language based on the pi-calculus, *in* G Plotkin, C Stirling & M Tofte, eds, 'Proof, Language and Interaction Essays in Honour of Robin Milner', MIT Press

Sangiorgi, D & Walker, D (2001), *The π-calculus A Theory of Mobile Processes*, Cambridge University Press

Schneier, B (1996), *Applied Cryptography*, Wiley, chapter 24 Example Implementations, p 576

Seibel, P (2005), *Practical Common Lisp* , APress

Thompson, S (1999), *Haskell The Craft of Functional Programming* , Addison Wesley

Wojciechowski, P & Sewell, P (1999), Nomadic pict Language and infrastructure design for mobile agents, *in* 'First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)', Palm Springs, CA, USA
*citeseer ist psu edu/wojciechowski00nomadic html