UNIVERSITY OF SOUTHAMPTON

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS

School of Electronics and Computer Science

A Multi-Agent Framework for Developing Complex Software

by

Layla Gordon

BSc (Hons) Computer Science with Artificial Intelligence

Thesis for the degree of Master of Philosophy

June 2007

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS

SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Master of Philosophy

A MULTI-AGENT FRAMEWORK FOR DEVELOPING COMPLEX SOFTWARE

by Layla Gordon

Human Biometrics, with gait as an example, has been around as a science for several years. Applications of the gait biometric can be found in the fields of medical diagnosis, physical therapy and sports. However, it is only in recent times that its use for human identification has become practical for real life and real time applications due to the increase in commercially available computing power.

The availability of a suitably powerful computing platform is only one aspect of building such system, there still remains the question of how to realise its software architecture. Without a viable solution to this problem the application and use of gait identification in a real life scenario will remain in its infancy.

Such a system is complex in two respects. One is the complexity in the computations of its individual components and the other is the complexity in building the layout of these components and communications between them.

Additionally there is the difficult task of efficiently maintaining such a system. If the framework that holds the components is not flexible enough, adding new components will be an extremely expensive and difficult task.

Therefore, this work focuses upon the creation of a general-purpose, multi-agent framework. As such, the primary goal of this architecture is to reduce system complexity allowing ease of maintenance and expansion, while also allowing distribution of data processing.

This framework is entirely general in its application and can be used for holding any software system capable of distributed execution. By using gait identification in this work as an example application, the framework demonstrates its capabilities and thus the advantages of using it.

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my supervisors John Carter and Adam Prugel-Bennet for their help, enthusiasm and friendship during this research. Special thanks are also due to Dr. Richard French and Andrew Gordon for technical advice and support. Also, many thanks to Department of Computer Science and Ordnance Survey for their help and use of equipment. This work was supported by a studentship from DARPA HumanID project carried out in ISIS. I also gratefully acknowledge the partial support by the European Research Office of the US Army under Contract No.N68171-01-C-9002. This work is dedicated to my family both in UK and Iran and all the friends I have made during my studies.

Chapter 1

Introduction

The main objective of this work is to present a solution for the design and realisation of complex software systems. This solution is a light-weight multi-agent system framework. Complex software systems exist in both industrial and academic environments.

As described in [31], in a typical complex software systems, there are many parts which have many interactions. The complexity in such system however is not accidental but it is a property of such a large system. Therefore, software engineering rules and paradigms are needed for providing a structure which makes it easier to handle such complexity.

The complex problem to be solved in this work is a biometric identification system.

Human biometrics, with gait as an example, has been around as a science for several years now. However, it is only in recent times that its use for identification has become practical for the real world, real time applications.

There are two main reasons why a gait recognition system is complex in nature:

- The algorithms involved are computationally intensive.
- There are multiple types and levels of interactions amongst these algorithms.

Using gait as an identification system would generally happen in a scenario where a query is made to the system and the result of identification is needed almost instantly. This could, for example, happen at an airport for validating the identity of a person backed up by their passport.

Considering the main characteristics of this system mentioned in the above bullet points, an appropriate software design is needed to optimise the run-time of the algorithms and minimise the complexity of interactions between them, otherwise, applying a gait identification system to a real world scenario would still remain in its infancy. Even in an off-line system where the running time of the algorithms would not be a critical issue, the complexity of interactions could make the addition of new components a very difficult task.

Therefore, an agent-based framework is proposed with the aim of solving the problems addressed above by reducing the complexity of interactions and distribution of data processing for improving the speed. Applications of gait as a biometric can be found in the fields of human identification, medical diagnosis, physical therapy and sports but as a part of this thesis we will be mainly focusing on demonstrating algorithms that produce gait signatures for human identification.

1.1 Goals

The main goal to be met by this work is an easily maintainable framework which allows for the *rapid* and *easy* design and development of a complex system.

As mentioned in the previous section, such a system is complex in two aspects. One is the complexity in the computations of its individual components and the other is the complexity in building the layout of these components and communications between them.

The second issue addresses the difficult task of efficiently maintaining such a system. If the framework that holds the components is not flexible enough, adding new components will be an extremely expensive and difficult task.

As mentioned in the previous section, the application which will be developed using this framework is human motion (gait) identification. Input data to this system is a video file of a subject's walk and the user is given the choice of using either the whole of this video or a segment of it. The next stage is for the user to select and chain any number of algorithms together to be applied to the data. These algorithms range from various methods for pre-processing the video to retrieving the signature of the subject's walk. They would then be provided with some form of meaningful feedback or output. The main goal here is for the system to be easy to use and to provide the necessary support for the addition of new algorithms by different users. To prove the concept, this system will be developed for use amongst the researchers in the gait identification research group. This example is an ideal platform since there are *many* researchers developing many different algorithms. Therefore, the flexibility of the systems' framework becomes an important priority. The greater the flexibility, the easier it would be for the researchers to add new algorithms or replace the old ones without the need for making major modifications to the underlying framework.

1.2 Gait Identification Applications

The above gait identification system can be divided into two major flavours. The first one is a single-run demonstrator which is capable of processing one video at a time. The second one is a batch-run which is designed to process large numbers of videos as quickly as possible. The workload can be highly distributed over a cluster of high-spec machines. The modular nature of this framework allows for easy distribution of the workload simply by starting more than one instance of a certain component.

Both of the above flavours share a number of common components, such as computer vision algorithms or pre-processing of the images. Therefore the major difference is in the end user and visualisation components.

Both of these scenarios are looked at in more detail in the next two sections and the ways each one benefits from the proposed flexible architecture is discussed in other future parts of this document.

1.2.1 Multi Component Gait Identification Demonstrator

As discussed in 1.2 this is one of the flavours of a gait identification application.

The main goal to be met by it is to be easy to use and to allow for a simple demonstration of gait algorithms. One of the main characteristics of this application is the demand for a framework to handle the dynamic nature of its architecture. This application is dynamic for one main reason. Throughout the life of this research, the researchers will be developing new

algorithms. Therefore there is more or less a constant need for adding new algorithms or replacing the old ones which make up the demonstrator. This demonstrator acts as a front end and a gateway to the world of research which happens behind the scenes. Therefore, it needs to be updated very regularly to highlight the state of the art of certain research, in this example gait.

It is obvious therefore, that it would be very beneficial for this application to be easily maintainable. This means that the addition and deletion of components (algorithms) can be done using an easy plug-and-play manner as opposed to the need for re-writing the framework for accommodating the changes.

One example is instead of having fixed menu items for the available components, these menus can be created on-the-fly from a list of components which are active at the time in the environment. The status of these components in the menu is refreshed in real-time depending on whether they are active or in-active. Insertions and deletions can also be done to the menu items in a similar fashion. For example, the image processing menu could have anything from one single item (Silhouette Generator) to many and as soon as any of these components are shut down, the menus are also refreshed.

The other issue is to allow for a *clear* separation between the data processing and the GUI. This is achieved by the highly modular nature of this framework by having a *single* GUI agent and multiple data processing ones. Figure 1.1 shows the simple architecture of this multi-component demonstrator. Here, the router agent plays a central role in knowing what agents exist in the system and also provides methods for the agents to contact each other. In the example given in Figure 1.1, each of the six agents in the system is representative of various processing stages needed for performing gait identification algorithms. Each one is called by the gait demonstrator application as and when needed and this is all done through the router.

A screen shot of this multi-component gait demonstrator is shown in Figure 1.2. This program allows the user to open a video file for gait processing. The buttons at the bottom of the GUI marked (Prev, Play, Stop, Next) allow the user to browse through the video. There are buttons to mark the entering and the exit frames of the subject in the video and also to step to the "in" and "out" frame marked by the user. The Filters Menu options at the top of the GUI would then allow the user to apply various algorithms needed for processing the video for retrieving a gait signature from the subject in the video.

1.2.2 Multi Component Batch Video Processing System

This is the second flavour out of the two applications mentioned in 1.2. The main aim of this application is to allow the user to process large numbers of video files with any number and combination of gait and image processing algorithms. Most of these algorithms are highly computationally intensive. Therefore, the immediate need is for the proposed framework to vastly reduce the time required for processing the data.

If each algorithm is wrapped inside an individual module and it can run independently of another, a framework can be designed to allow them to run in parallel as opposed to serial. This is also known as a divide-and-conquer strategy and modular systems can easily benefit from the advantages of this strategy.

Therefore, to cut back the processing time, this framework will run on a high performance compute cluster which also benefits from a transparent file system to allow resource sharing. The transparent file system is a name space which is the same across the cluster and it is accessible via the utilities of all connected machines. This file system is mounted on all the

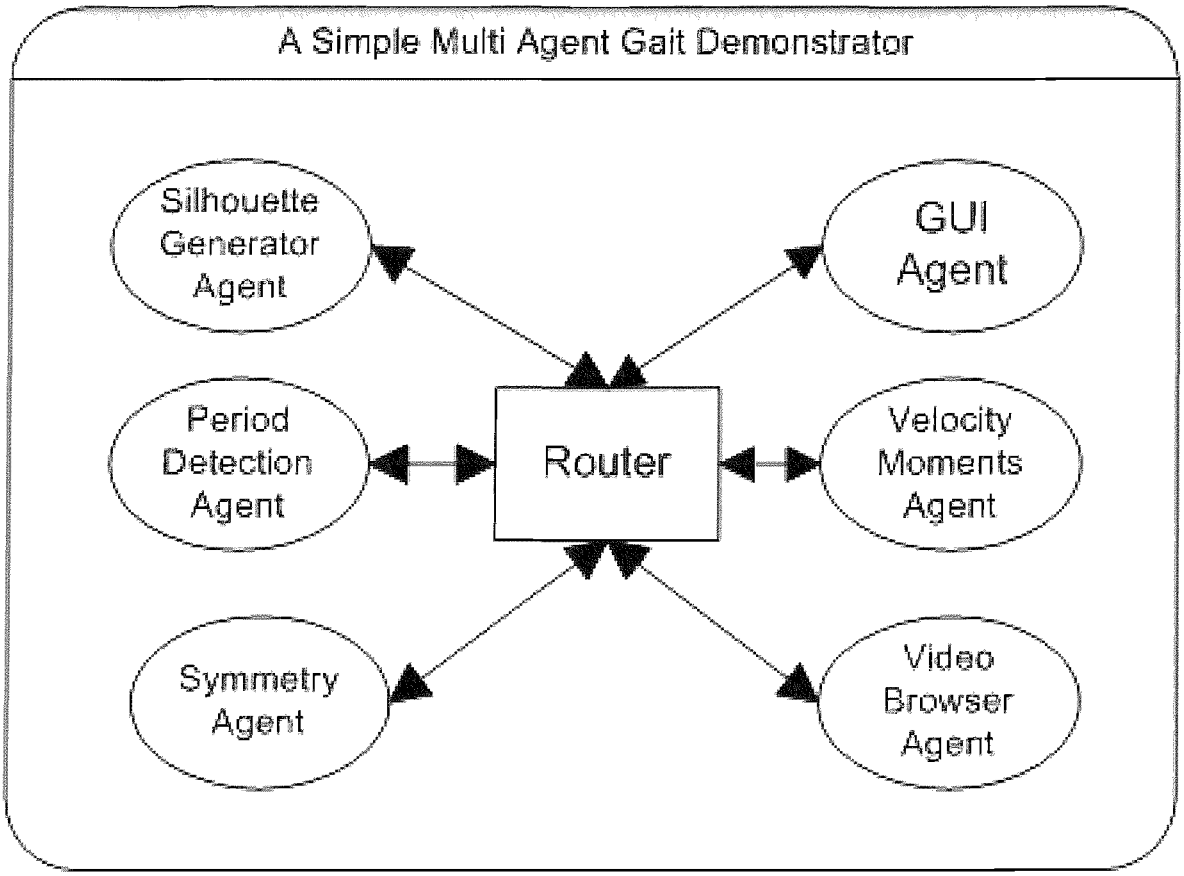

Figure 1.1: A multi-component gait demonstrator

machines in the cluster allowing all to see and access files on the transparent file system through the same directory scheme.

In a simple scenario, more algorithm components are started up for data processing as and when the workload is increased.

1.3 Motivation

With both applications mentioned above in mind, an earlier attempt was made to develop a system which encompassed both applications of the gait identification. This system met the goals, however, its disadvantages and problems started to manifest themselves when it was put through a series of tests. These tests challenged the system in the two important areas of *scalability* and *re-usability*.

A screen shot of this system has been provided in Figure 1.3.

Due to the nature of the gait identification research, the first and most important aim at the time was to develop this system rapidly and for it to work in a simple way. There were important deadlines to meet which meant that large data volumes had to be processed and therefore very little time was spent on the design phase of the system. Most of the effort was spent on the system's development. This resulted in the system being monolithic and very hard to scale and maintain.

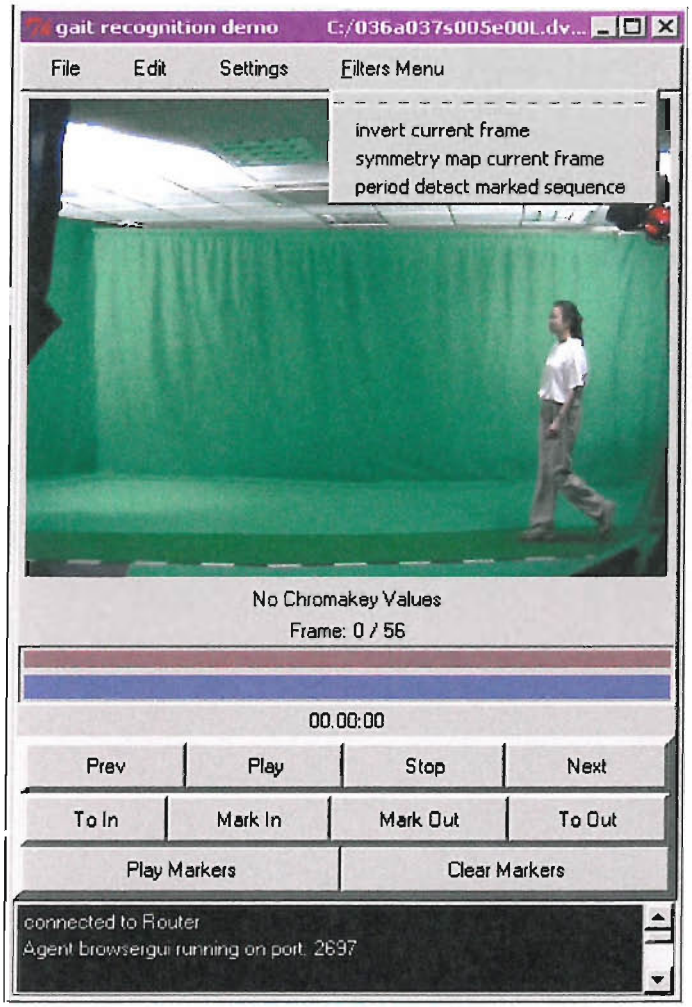

Figure 1.2: A screen shot of the Multi-component gait demonstrator program

In the next two sections, the scalability and re-usability are looked at in a bit more detail in the context of the first generation gait identification system.

1.3.1 Scalability

The first generation gait identification system was only capable of processing a single job at a time and there was no facility for performing batch jobs. It also could not be run on a cluster of machines. The use of a cluster of machines in a parallel manner is a highly important characteristic since, as mentioned previously, the gait algorithms are computationally intensive. Therefore, the importance of using parallel processing and distributed computing is clear. These two aspects are required in this work, since, the parallel processing of a program's components improves computational efficiency by distributing the work load. Utilising a cluster of machines using the distributed computing concept means the system will be scalable and be used for processing anything from a small to a large number of jobs.

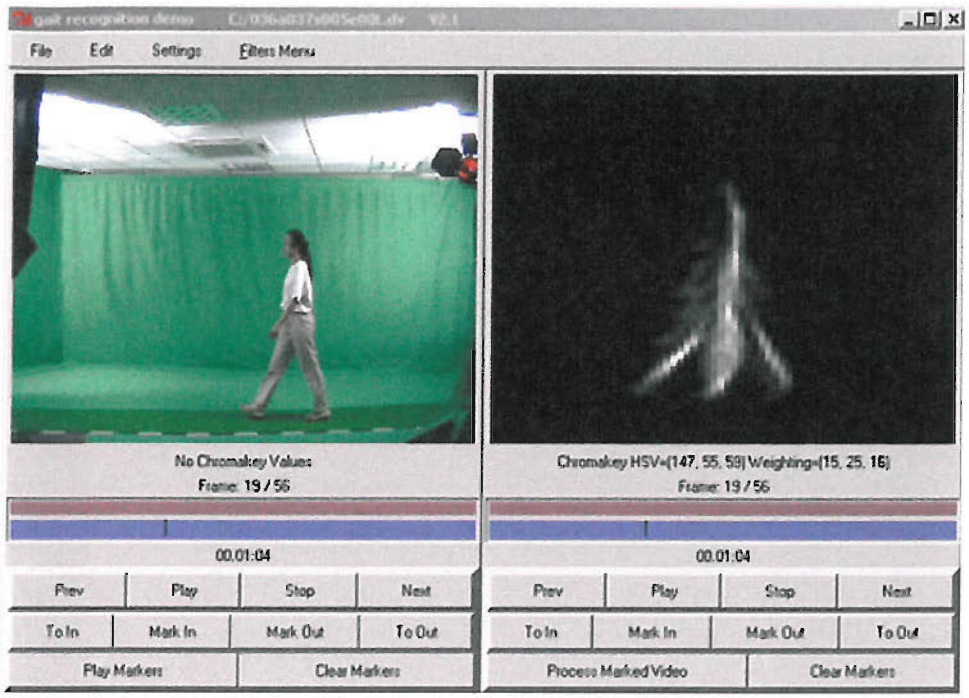

Figure 1.3: A screen shot of the old Gait Identification system

1.3.2 Re-usability

As mentioned in previous sections, very little time was spent on the design, therefore, the gait recognition and image processing algorithms were hard-coded into the system. It was almost impossible to modify the system with addition or deletion of algorithms without the need for a major re-write of the code for the structure.

Additionally, the user interface (menus and commands) were fixed and hard-coded. This means that no support existed for dynamic addition of menu items when new algorithms became active in the system.

Re-usability and maintenance are very important attributes when it comes to a system which aims to demonstrate the algorithms developed in a research team. Old algorithms go out of scope and new ones are added throughout the life cycle of the research. Therefore, any proposed system would need to benefit from easy maintenance.

There were of course other problems too, however, the two above mainly indicated the need for the change of framework to a more flexible one.

Therefore, since this is a framework re-design issue, a number of different software engineering paradigms were reviewed to assess their suitability. The paradigms chosen are appropriate to the project. For example, agent oriented vs. traditional distributed computing are two different alternatives from the framework architectural design point of view and object-oriented vs. extreme programming (XP) present the alternatives for the design and implementation of an application framework. These different paradigms are discussed in greater depth in the following chapter.

Chapter 2

A Brief Review of Software Development Paradigms

When it comes to solving the problem outlined in the previous chapter, the first thing that comes to mind is to review the existing solutions. Since this is a software engineering problem, in terms of providing a suitable framework for solving complex systems, it is appropriate to look at a few software engineering paradigms which are suitable for the design and realisation of such framework. In this chapter, a brief description of software paradigms is given and then a conclusion is made at the end stating which one is best suited to this work. It is important to do this review is so that a methodology can be derived based on existing solutions for solving the complex system outlined in Chapter 1.

Software development as an activity can be very chaotic. In a simplistic manner, software would be written with no or very little design in advance. Thus, the problems are fixed at the end when the user tests the system. This type of development works in cases where the project is very small in scale, but when such a system is about to grow and new features are to be added to it, problems arise.

Software engineering paradigms were first introduced in the 1980s to solve this problem. The role of these paradigms is to introduce disciplined set of processes during the development of software so that this phase would become more efficient and more predictable. There are a set of rules to follow and set of documents to produce all the way throughout the project life cycle. The rules introduce ways of representing and modeling various parts of the software system and the relationships between them.

Some of these methodologies proved to be inefficient in that following the highly disciplined rules was a very time consuming process. This resulted in slowing down the project development which made using these methodologies extremely cost ineffective. However, the benefits gained from using a software development methodology resulted in researchers moderating the rule sets and developing new methodologies which still have all the benefits of following a disciplined approach, but have fewer drawbacks in terms of time and project resources.

This new group of methodologies has been labelled as Agile (e.g. lightweight). The appeal of this new generation of paradigms is due to having lightweight rule sets which means that the agile software paradigms are:

- less prescriptive: The lighter rule sets results in less frustration for the programmers. It also cuts down the redundant time spent on following the unnecessary rules.

- less bureaucratic: Large cut-down on the rules that are rigidly devoted to the details of administrative procedure.

The user of the agile paradigms will find a compromise between having no rule sets and too many rules, providing just enough to reflect a desirable payoff. Below, a few popular software engineering methodologies have been described along with a brief description and comparison of alternative distributed computing frameworks. It has also been shown how design of the applications discussed in the Chapter 1 (see Section 1.2) would benefit from these.

These methodologies can be classed into two main groups:

- Design and Implementation: Extreme Programming and Object-Oriented.
- Framework: Agent-oriented and Traditional Distributed computing

The second group is reviewed for the purpose of finding a suitable methodology for the design of the framework. The agent-oriented methodology is then compared against traditional distributed design to see which one would be more suitable for the structure of the framework needed in developing the complex gait identification system. The first class of methodologies, extreme programming and object-oriented, however, are methodologies needed for design and implementing the actual system once the architecture of the framework has been decided on. These will help model the various parts of the system and develop the code. Hence, in the conclusion of this chapter, the comparisons are made between the two concepts in class one and then class two but not across classes.

2.1 Object-Oriented Programming

The idea in this paradigm is that if a methodology is used which allows us to *identify* and *classify* the components in our software world the same way as we do in natural work, then, developing this software would become easier to understand, follow and evolve.

Below are a few fundamental concepts that surround the object-oriented paradigm.

2.1.1 Concepts

Objects and Object Classes

In the object-oriented paradigm, the problem domain is modelled using object-classes, and their instances [4]. An object is any abstract or real world item that is relevant to the system. And a class is a definition of an object. Our natural world is a collection of collaborating objects, therefore, we can organise software according to the structure of the natural world. For example, in a supermarket our objects can be things like milk, apple, bread and so our classes would be dairy, fruit and bakery.

Objects are comprised of attributes, operations and behaviour.

- *Attributes*: The local data that describe the object.
- *Methods*: Operate on an object's attributes and give access to them or change their values.
- *Behaviour*: The objects receive events which change their attributes and more events are sent out as results.

Methods

"Methods specify the way in which an object's data are manipulated" [36]. Looking at the example given in the previous section, the method for object milk would be purchase-milk.

Encapsulation

This is one of the key features of this paradigm. Encapsulation means keeping the object's attributes and methods together. Hence attributes can only be accessed via methods. "The object hides its data from other objects and allows the data to be accessed via its own methods" [36]. For example, Milk-price is an attribute of object Milk which is accessed via Milk-purchase method.

Inheritance

In this paradigm, high level objects are defined, together with lower level objects which inherit some of their properties from the high level ones. The notion of super class is defined which is the class that the current class is inherited from. R. Wirfs-Brock, B. Wilkerson and L. Wiener in [55] define inheritance as a classes ability to use a superset of definitions of a single class or a group of classes to define the data structure and behaviour of its instances.

Multiple Inheritance

This concept is very similar to the single inheritance with one difference in that a lower level subclass may inherit properties from more than one super class.

Polymorphism and Dynamic Binding

Polymorphism allows the same object to react to different forms of the same method. For example if an object "shape" exists with a method "draw", depending on the call, the draw method might refer to drawing a rectangle or a triangle. Therefore, the object type does need to be known in advance and the correct implementation of the same method gets called at run time. This is called dynamic binding.

The main advantage of Polymorphism is the extensibility of the code written. Abstract interfaces can be written for objects which can be re-used as and when new objects are added to the system.

2.1.2 Benefits

- *Naturalness of Analysis and Design (Cognition)*:

  As described in [37] and [48], one of the frequently mentioned benefits of the object-oriented paradigm is the assumption that is cognitively similar to humans understanding and perception of real world. This therefore means that it should be natural for software developers and users using this paradigm to map the problem into objects and into classification hierarchies.

- *Software Reuse*:

  Many researchers of the object-oriented paradigm claim that this paradigm provides effective mechanisms that allow the easy reuse of the developed software. See [37].

T. A. Budd in [8] states: "Well designed objects in object-oriented systems are the basis for systems to be assembled largely from reusable modules, leading to higher productivity". J. Martin and J. Odell in [36] similarly state "It, 'OO', leads to a world of reusable classes, where much of the software construction process will be the assembly of existing well-proved classes".

The encapsulation property of this paradigm allows new objects to be introduced and used in the system without the need to modify any other part of the software. Therefore, in an ideal world there would be some standard top level super classes available to be purchased and inherited from to make them suitable for being used in any scenario.

- *Communication Process*:

  B. Curtis in [11] as well as H. Krasner, B. Curtis and N. Iscoe in [34] report that at the software team level, some of the key problems encountered are communication and coordination, capturing and using domain knowledge and organisational issues. With the object-oriented approach, advocates claim that communication and coordination between the project team and client(s), and also within the team are enhanced.

- *Refinement and Extensibility*:

  Object-oriented researchers also claim that software developed using the object-oriented paradigm is easy to refine and extend. S. Khoshafian [33] states "Object oriented programming techniques allow the development of extensible and reusable modules". On a similar note in [22] we have "Inheritance, genericity or other forms of polymorphisms make exception handling easier and improve the extensibility of systems". These claims are related to three key principles of the object-oriented paradigm which are encapsulation, inheritance and polymorphism.

- *Decoupled modules*: The objects developed in this paradigm are decoupled and can be modified independently.

2.1.3 Problems

- *Slow development*: It is very important in object-oriented design to come up with an optimised breakdown of the problem domain into modular components. This means deciding about the correct top-down object hierarchy, inheritance, object attributes and others which were described in Section 2.1.1. Therefore, the design stage can take up a large amount of time.

- *Little work on problem investigation and discovery*: Due to the highly philosophical and conceptual nature of this paradigm, more time is spent on program structure and less time on the actual program functionality.

2.2 Extreme Programming Paradigm

A broad definition of extreme programming (XP) is given by R. Jeffries in [30]. He states that this paradigm is a set of disciplines based on simplicity, communication, feedback and courage for developing the software. The main emphasis here is to bring a whole team together using simple practices and also with enough and regular feedback to enable such team to observe

which stage of development they are in and to be able adjust their methodologies and practices to suit unique situations which may arise during the development.

This paradigm is a unique blend of deliberate and disciplined approach to the software development. One of the main focuses of this methodology is customer satisfaction.

Therefore, the main aim is to deliver the software the end user needs and when it is needed. This presents the developers the power to be able to respond to the changing requirements of the customer as it arises. Even if this occurs very late in the life cycle of the development.

2.2.1 Core Practices in Extreme Programming

There are a few practices that should be followed within the remit of the extreme programming methodology. These are as follows:

Whole team

An important issue in this paradigm is the team structure. The concept of "The Customer" is introduced which applies to the representative from the business. The customer is involved in the daily activities of the team and sits with them through out the day. All the contributors to the project is an integral part of the "whole team".

Planning game, small releases and customer tests

Few simple concepts such as planning and tracking are used by the team to plan the future of the project and what should be done next. Based on the rules that the customer has defined, the team also produces a series of small fully-integrated releases which are focused on their business value.

Simple design, test driven development and design improvement

Design in this paradigm is very simple and by thoroughly testing the code all the way through out the project, the design is constantly improved to always allow it to reflect the current needs.

Continuous integration, collective code ownership, pair programming and coding standard

The core systems are integrated continuously and are kept in a running state at all times by the team. The code is written by pairs of programmers but they also all work together. The style chosen for coding is usually a consistent one amongst all the programmers, so that everyone in the team fully understands it. This enables all the members of the team to update and modify all the code as and when needed.

Metaphor and sustainable pace

The pace at which every member of the team works is very important in this paradigm. Therefore, it remains a necessity that everyone works at a pace which can be maintained in the long term. Another important concept is for everyone in the team to share a common and simple picture of what the system looks like [30].

2.2.2 Benefits

As mentioned before and discussed in [30], using this paradigm, software development improves in three main areas which are *communication*, *simplicity*, and *Courage*.

- *Immediate feedback to the programmer through tests*: One of the most striking characteristics of extreme programming is the strong emphasis on the testing of the software. Other paradigms put a very low emphasis on testing compared to the extreme programming. In extreme programming testing is placed at the foundation of development. Every programmer would need to write a test suite for every part of the software that they produce. These tests are all integrated into processes which will result in a very stable platform for developments in future. Such tests are also beneficial to the programmers themselves in providing them with instant feedback throughout the development life-cycle.

- *Re-factoring yields design improvements*: XP builds an evolutionary design process that relies on re-factoring a simple base system with every iteration. Re-factoring is the process of continuous design improvement. For more information on re-factoring see [17]. All design is centred on the current iteration with no design done for anticipated future needs. The result is a design process that combines discipline with adaptivity in a way that arguably makes it the most well developed of all software engineering paradigms. Re-factoring in XP is a bi-product of Courage. Courage is also apparent in two other areas introduced by XP which are where the programmer knows when to throw the code away as soon as it is obsolete and also to be persistent. This is when a whole day is spent on coding a problem and when the results are not achieved, the programmer carries on the next day and solves the same issue in no time. Therefore, courage is a very important characteristic in XP.

2.2.3 Problems

Very little time is spent at the design stage. This means that an end product may not benefit from properties such as re-usability.

2.3 Traditional Distributed Computing on PC Clusters

Using a cluster or network of workstations for processing computationally intensive jobs such as image processing is very popular. In this paradigm, different users who all have access to a cluster, run different programs on different sets of machines. However, in some cases one large distributed program can use the whole cluster to complete its task.

However, there are immediate problems that arise from running a traditional distributed program on a cluster:

- While one distributed program is using the cluster as a whole, it has no control of other users submitting jobs to the machines that it is running its tasks on. Hence, the processors within the cluster can become overloaded.

- Users that login to individual workstations on the cluster may demand an exclusive use of the workstation.

- Reports and logs need to be constantly generated and updated to provide the single distributed program with information for it to be able to re-distribute its tasks.
- The workload on each workstation is measured using the operating system parameters not just the size of the task within the program.

2.4 Multi-Agent System Paradigm

Multi-agent system is a new paradigm for developing software. Software agents are currently a major focus of many researchers with an interest in computer science and its relation to artificial intelligence.

An agent is an expert piece of computer software which can perform autonomous actions in order to meet the objectives of its design. It is very difficult to describe the concept of autonomy but the simple meaning in this context is that an agent will be able to perform an action without a direct intervention of humans or other agents. It will also have a full control over its internal state and performed actions. Due to obvious similarities, it is possible to draw an analogy between the multi-agents' concept and one of object-oriented systems.

In an object-oriented system, an object holds control over its state. This state can only be viewed or changed via object's own methods. In multi-agent systems, an agent similarly encapsulates a state. But additional to state, they also encapsulate behaviours. This does not apply to objects since they have no control over their methods' execution. If an object "a" invokes methods "m" on an object "b", then object "b" would have no control or choice about whether the method "m" is performed on it or not. This means that in this example, since object "b" holds no control over its actions it is therefore not autonomous.

On the other hand, an agent is expected to have exactly this kind of control over its actions. As mentioned in [32], because of such distinction, an agent is not thought of as an entity which invokes actions on other agents. Rather, an agent requests actions to be performed on the recipient agent. The choice remains with the recipient for the action to be performed on it or not.

The most obvious benefit gained from using this paradigm is modularity. The concept of an agent to represent a particular component enforces a system to yield itself to a modular design. The communications between the agents also means that they can run on separate machines and as long as these machines are networked, the collaborative method of performing the task can be achieved.

However, one of the immediate downfalls here in this paradigm is the extra overheads needed for the design of such multi-agent system. Trade-offs need be to be assessed carefully to determine the balance of benefit gained from using such paradigm versus the costs involved in the design phase. The other issue is communication. Reliable network and transparent file systems are essential if full advantage is to be gained from running a multi-agent system on a parallel and distributed computer architecture.

2.5 Conclusion

In this section, we perform a comparison of all the paradigms mentioned in the previous section and outline the advantages and disadvantages of these relative to this project. As mentioned

in 1.1, the main goal of this project is to build an extensible and easily maintainable framework for developing a complex system such as gait identification.

The software development paradigms which were looked at in the previous sections fall into two main categories:

- Design and Implementation
- Architectural and framework

The agent-oriented and distributed computing approaches are potential applications developed using extreme programming and object-oriented software engineering disciplines. Therefore, it is only meaningful to compare and contrast software engineering disciplines as opposed to comparing, for example, multi-agent system and object-oriented approaches which are not in the same class. In many ways extreme programming and object-oriented approaches are not exactly in the same category either, but the reason they are compared is that the comparison is drawn on the of length of time and effort spent at the design stage. The opposite to the extreme programming in a direct comparison would have been an approach like IBM's Rational Unified Process (RUP). Contrary to extreme programming, in RUP, much emphasis is given to the requirements and design modelling. This introduces many overheads. There are also many rules to follow which need to be cut down. However, some of extreme programming pioneers have suggested that extreme programming is a subset of RUP.

2.5.1 Traditional Distributed Computing vs. Multi-Agent Systems

- In a multi-agent system, agents are primarily concerned with their own welfare whilst in traditional distributed computing, all the computing elements are assumed to share a common goal.
- In the presence of an open environment with dynamically changing components, a multi-agent paradigm introduces a few new and interesting concepts. These are mainly the result of this paradigm encouraging modular and extensive designs for the system components. The multi-agent paradigm can even be an underlying concept for understanding distributed systems. It is of particular importance if adaptability and flexibility are among the key requirements of the software application.
- The nature of multi-agent paradigm allows for developing applications which are *modular* and *distributed*, making it one of the best suited frameworks for developing the desired gait identification demonstrator software.
- In traditional distributed systems, there is a need for an extra component to manage load balancing across the varying systems. However, in the agent framework, the jobs are distributed as soon as agents become available and as such load balancing is implicit in the system design.

2.5.2 Extreme Programming vs. the Object-Oriented Software Development Paradigm

- *expensive design phase*: Multi-agent system can be difficult to build due to concepts such as autonomous behaviours, flexibility and being an individual entity. If such system

was to be developed using object-oriented methodology, the design phase would include disciplines such as role analysis and services design. As a result, the object-oriented paradigm would be expensive in this case where requirements may be weakly specified.

- *unclear end product requirements*: The end product's requirements in this scenario are highly coupled with the outcome of the developed program in the middle steps of the project lifestyle. Therefore, not having a very clear view of what the end product should be like, designing objects that might have to be changed in later phases will not be very cost and or time effective.

- *design phase overhead:* Since the environment that the project needs to be implemented is a scientific one, spending too long on a design phase can have the effect of an overhead.

2.5.3 Proposed Paradigm

Therefore, we can conclude that the paradigm proposed to use in this work is not going to be strictly one of the above mentioned ones, but will be a combination of a few. From the framework point of view, the approach is going to be a modular multi-agent system as opposed to a centralized one due to following reasons:

- A multi-agent system benefits from distributing hardware and software capabilities and computational resources across a network of inter-connected agents. For example in some scenarios in which a centralised system could easily suffer from limitation of resources, bottlenecks or critical failures, a multi-agent system would suffer very little due to being de-centralised and not having a "single point of control".

- In a multi-agent system, an existing legacy system can be interconnected and carry on inter-operating. The only necessary change would be adding an agent wrapper to the underlying framework to allow the various existing components to be connected to form an agent society.

- Given a complex system such as one in this work, sub-systems (e.g. various algorithms) will work together to achieve the functionality of the overall system. Each sub-system or component is aimed to achieve one or more objective. These components are therefore active and autonomous. This means that a very natural way to modularize a complex system to a series of components which acts and interact in many ways to achieve their goals will be achieved by using a multi-agent system concept.

  At any given level, subsystems work together to achieve the functionality of their parent system. Each component can be thought as achieving one or more objectives. Thus, entities should have their own thread of control (i.e. they should be active), and they should have control over their own actions (i.e. they should be autonomous). Given this fact, it is apparent that the natural way to modularize a complex system is in terms of multiple autonomous components that act and interact in flexible ways to achieve their objectives. Therefore, the agent-oriented approach is simply the best fit.

- In a multi-agent system, the agents can be in different physical locations. If the data is distributed across multiple machines, each agent can either process the data at its source or arrange for co-ordination of gathering various parts of data. Therefore, a multi-agent

system allows for co-ordination of data which may exist globally across various sources and provides methods for efficient filtering and retrieval of such data.

- Apart from the distributed data, a multi-agent system will also provide solutions in situations where expertise is distributed spatially and temporally.

- A multi-agent system enhances the overall performance of the system in areas of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility and reuse. Multi-agent systems yield computational efficiency due to providing the ability to distribute processing amongst several machines. They provide reliability, extensibility, flexibility due to the modular component-oriented design. Agents can be removed and replaced with new ones without the need to re-program the application. Hence, the resulting system would be more maintainable and easier to re-use. They are also very responsive due to agents being task-oriented.

From the implementation point of view a combination of extreme programming and object-oriented will be used due to the following reasons:

- Object-oriented concepts such as *inheritance* and *polymorphism* allow for all the communication handling to be handles in the base class, leaving the inherited classes easy to add. This means to add new agents to the system, the developer would only need to focus on new agent's functionality as all the communication will be encapsulated in the base agent class.

- The rapid development of extreme programming is very well favoured in the scenario to allow for quickly developed end products.

- Extreme programming (XP) allows for the design to be modified throughout the development. This allows to accommodate any change of strategy according to the feedback gained from the user, which in this case, is the developer.

The diagram in Figure 2.1 shows how the proposed paradigm lies in the plane of all paradigms reviewed:

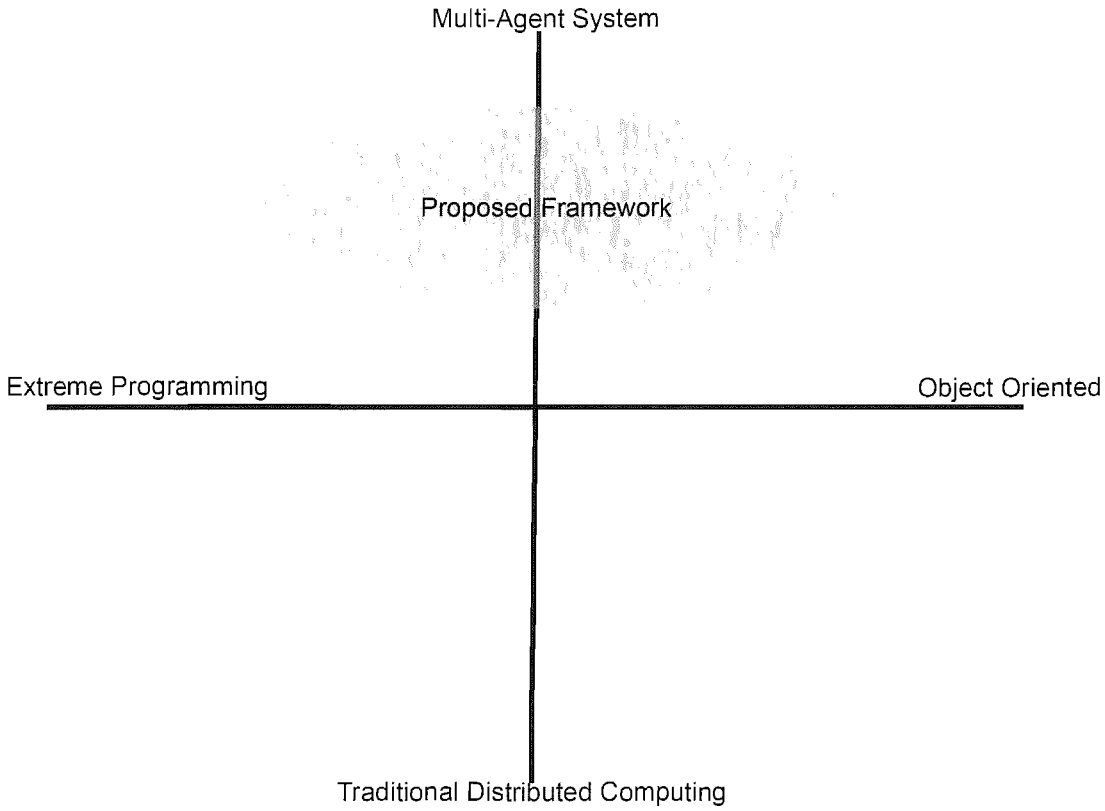

Figure 2.1: A diagram showing how the proposed framework lies in the plane of all other paradigms reviewed

Chapter 3

Agents and Multi-agent Systems' Technology

The last chapter was concluded by justifying why the multi-agent system paradigm is suitable for developing the complex system to be used for gait identification. Now that it has become clear what the solution is, there is a need to go into more detail about the multi-agent system paradigm. This is to explain in more detail what the term "agent" means in both an Artificial Intelligence (AI) and a software engineering context.

It is important to note that although the main focus is in terms of solving a complex software using agents, the reason it is important to include the AI definition is that a lot of agent attributes are borrowed from the AI usage of this term. Therefore, it is important to define the term agent' in both contexts.

This chapter continues with a background review of multi-agent systems (MAS) in software engineering and concepts used within this paradigm. Once a background has been given for the terms "agent" and "multi-agent systems", the focus of attention is moved to reviewing existing tools for developing agents and multi-agent systems.

Hence, this review is limited to the development tools which are used for developing complex software systems using multi-agent frameworks. This is important because it explains the decision to design a light-weight multi-agent framework with borrowed ideas from existing tools rather than strictly using one of the existing ones. This then forms the background information needed for design decisions made in the following chapter for the multi-agent framework produced for the problem discussed in Section 1.1.

3.1 What Is an Agent?

It is best to start with the definition of the term "agent". Straight from the Penguin English dictionary the term agent means "a person who is authorised to act for or in the place of another". However, in the world of Information Technology (IT) there is no definition for the term agent which is accepted universally. This results in much debate and controversial discussions surrounding the definition of an agent and multi-agent systems.

Essentially, while there is a general consensus that for example *autonomy* is central to the notion of agency, there is little agreement beyond this.

Most of the difficulty in arriving at a universal definition for agents and agent systems

is that the different attributes seem to have more or less importance when one moves from one application domain to the next. For example, in an air traffic control application, the prospect of such a system being able to modify its behaviour could be a horrifying thought. Whilst, this attribute is more important and even essential in other domains for example e-bidding or agents used in auctions.

Various definitions for the term agent exist in the literature. A selection is provided below which presents agents of different behaviours. In each below examples, some aspects of the term "agency" are more important than others. Hence, each has different performance characteristics and are used in accordance to the target systems' needs.

3.1.1 The WooldridgeJennings Agent

For example, the definition for an agent in [58] is:

"An *agent* is a computer system that is situated in some *environment*, and that is capable of *autonomous* action in this environment in order to meet its design objectives"[58].

On the other hand in [57] (page 2) there is more elaboration on the properties of agents:

- *Autonomy*: "Agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state"[57].
- *Social ability*: "Agents interact with other agents (and possibly humans) via some kind of agent-communication language"[57].
- *Reactivity*: "Agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it"[57].
- *Pro-activeness*: "Agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative"[57].

Although Wooldridge-Jennings' second definition highlights the three main properties of *autonomy*, *sensing* and *acting*, it can be applied to a broad but finite range of environments.

3.1.2 The Artificial Intelligence (AI) Agent

This type of agent as mentioned in [49] is a software systems which perceives the environment through its sensors and acts upon the environment through its effectors. The text book Artificial Intelligence: A Modern Approach([49]) was a very successful one and in 1995 it was used in 200 colleges and universities. It is clear that the type of software agents in this textbook are ones which embody Artificial Intelligence (AI) techniques.

The key notions such as environment, sensing and acting in the above definition are very generic ones. Therefore, depending on how you define and restrict them, the system which takes input, performs action and produces output could easily be interpreted as any ordinary

software program. This shows that such key notions should be very precisely defined to characterise an agent as opposed to any other conventional software program.

3.1.3 Maes Agent

P. Maes (see[35]) of MIT's Media Lab, is one of the pioneers of agent research. She adds a crucial element to her definition of an agent: agents must act autonomously so as to "realize a set of goals". Also environments are restricted to being complex and dynamic. It is not clear whether this rules out simple systems such as a payroll software system.

3.1.4 The KidSim Agent

In [52], the authors who at the time of writing this were working with Apple Inc. define an agent as a "persistent software entity" which is dedicated to a specific task or purpose. The distinguishing characteristic here between agents and sub-routines is revealed here is for the entity to be "persistent". Agents possess the knowledge of how to perform their tasks and accomplish their agendas. They are also easily distinguished from multi-function applications due to each agent serving a "specific purpose". They are also typically smaller than other multi-function routines. One would need to consider however though many agents are "special purpose" but this is not an essential feature of being an agent.

3.1.5 The Hayes-Roth Agent

In [26], B. Hayes-Roth of Stanford's Knowledge Systems Laboratory states that the agents' reasoning ability is more active during the time they are selecting their actions. She believes that the intelligent agents perform three functions continuously. These are "perception of dynamic conditions in the environment, action to affect conditions in the environment and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions"[26].

If reasoning is interpreted in a broad manner, the agent architecture described here would allow for reflex actions as well as planned ones.

3.1.6 The IBM Agent

As described in [24] which is white paper from IBM's Intelligent Agent Strategy, "Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires"[24].

This definition assumes that an intelligent agent in this system can act for another agent provided authority is granted by the other one. There are eight possible example applications in this paper which a simple example of might be an information gathering agents.

Another influential white paper from IBM [21] describes intelligent agents in the shape of three dimensional space defined by notions of *agency*, *intelligence*, and *mobility*. This is shown in Figure 3.1.

Here [21] states: "Agency is the degree of autonomy and authority vested in the agent, and can be measured at least qualitatively by the nature of the interaction between the agents and other entities in the system. At a minimum, an agent must run asynchronously. The degree of agency is enhanced if an agent represents a user in some way. A more advanced agent can interact with data, applications, services or other agents" [21].

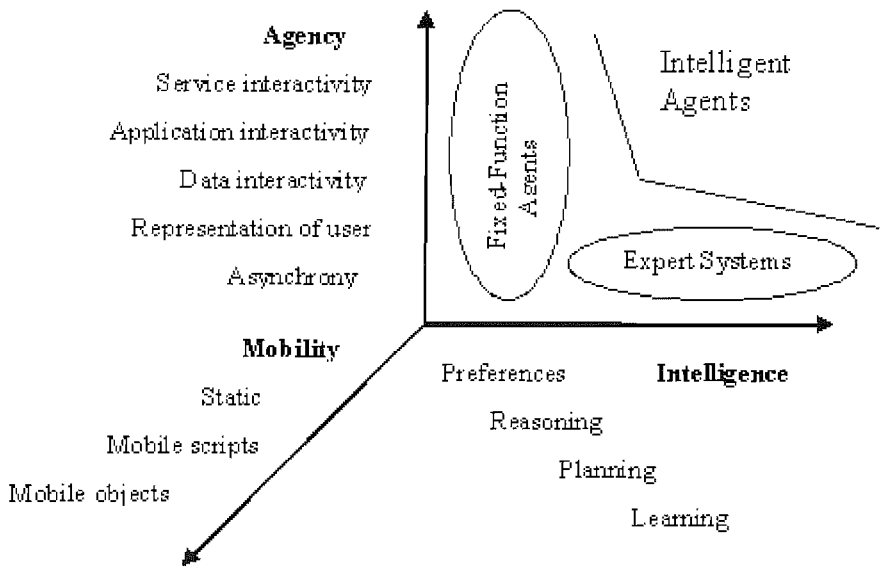

Figure 3.1: Scope of intelligent agents (Adapted from [21])

3.1.7 The SodaBot Agent

According to M. Coen [9] in MIT AI Lab,

> "Software agents are programs that engage in dialogs [and] negotiate and coordinate transfer of information." [9]

As we can see this definition is slightly different from the other ones mentioned since intelligence and mobility are not mentioned as part of the definition. However, to achieve the mentioned "negotiation" requires agents to be able to sense and act. Also, to achieve a good "dialogue" they would need to be able to have a great handle on communication. And these are the important criteria which make even this more lightweight definition different from standard non-agent software programs (See [9]).

3.1.8 The Foner Agent

Agents here in this definition are different to the ones in the last section. There are much more expectations for a program to qualify as an agent here. Foner agents in [15] collaborate with end-users to gain feedback and use it to improve the accomplishment of the users' task. Therefore, in addition to "autonomy" this goes a step further since the agent here would need to ensure a trust in its dialogue with the user and in the case of this trust degrading it would then need to be able to adapt to strategies for coping with a communication mismatch.

3.1.9 The Brustoloni Agent

As discussed in [6],

> "Autonomous agents are systems capable of autonomous, purposeful action in the real world."

Unlike some of the definitions from previous sections, the Brustoloni agents must live and act in "real world". Therefore, this definition instantly excludes standard software programs again and some other agent definitions seen above. He also insists on a new attribute for his agents and that is for them to be "reactive" which means that they are able to respond to asynchronous and external stimuli as they occur in real time.

After looking at all the above definitions for agents, below is a set of attributes which can be captured in Agents as described in [19] [13]:

- *Reactivity*: To be able to sense selectively and act upon it.
- *Autonomy*: To exhibit behaviour which is self started, goal-directed and pro-active.
- *Collaborative behaviour*: To work in harmony with other agents toward achieving a common goal.
- *Knowledge-level: communication ability*: To communicate with languages closer to human speech with the users and other agents rather than using machine-oriented and symbol-based protocols. [38]
- *Inferential capability*: Working beyond the acquired information, the agent will gain explicit models itself, the user, the environment, situation and other agents. If given an abstract specification for the task, the agent is able to infer a deeper level of specification using prior knowledge of previous goals it has achieved or preferred methods existing in its environment. This would result in more flexible agents.
- *Temporal continuity*: To maintain their state and identity consistent over very long periods of time.
- *Personality*: To be able to mimic and demonstrate human-like attributes in a believable manner. An example could be emotions.
- *Adaptivity*: The ability to learn from the experiences and environment to improve.
- *Mobility*: To be capable of migrating from source platform to another one in a self-directed way. This means without user-intervention and purely by agents decision to move and where to move to.

3.2 Taxonomy of Agents

Three major views of agency can be distinguished in the literature.

3.2.1 Programmer Agents

This class of agent refers to one used by computer scientists and software engineers as an abstraction to conceptualize, design and implement complex systems. Here the entity that benefits most from this class is the programmer.

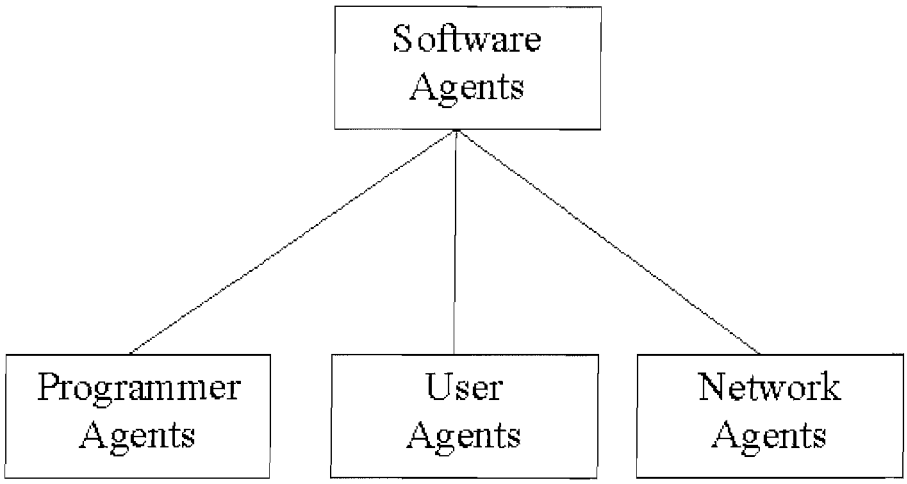

Figure 3.2: A simple taxonomy for software agents

3.2.2 Network Agents

These are a class of agent which act on behalf of network nodes by autonomously migrating in a distributed environment. Here the entity that benefits most from this class is the computer network.

3.2.3 User Agents

This class of agent defines an abstraction for the end users for interacting with computer systems. There the entity that benefits most from this class is the end-user.
The relation between these is shown in Figure 3.2.

3.3 Multi-Agent Systems

One definition for a multi-agent system is the following: A co-operative working environment comprising synergistic software components which can cope with complex problems.

There are three main types of communications which can occur between agents in a multi-agent system:

- Cooperative interaction
- Contract-based co-operation
- Negotiated cooperation

Contract-based cooperation and negotiation involves adjusting the terms of an agreement rather than following a more rigid set of rules such as in protocol-based negotiation. Agreements are reached amongst the agents and are regulated by contracts. These agreements can result in contracts to be adjusted according to involved agents internal theories.

3.4 Locating Agents

In an environment of more than one agent, to handle the complex interactions, there is a need for handling fusing, presenting, finding, advertising, updating and managing services for agents and the information. This resulted in introduction of a new concept known as "middle agents". [12]

The agents in the environment advertise their capabilities to the middle agents. These are very similar to estate agents in a way that they receive requests and adverts and try match one to another and put agents in touch with each other due to their demands. Therefore in each transaction which middle agents are involved in they play neither the role of the requester or the provider. They simply act as a middle man. The obvious advantage of using middle agents is that in a dynamic multi-agent environment, the existence of such agents would help with running the system robustly and smoothly when agents could appear, disappear or migrate in real time. (see [47])

Several definitions exist for types of agents which fall under the concept of *middle agents*. Some of these are described below, however, the definitions can be very vague and make it hard to enable one to clearly differentiate between them.

There are several types of agents that fall under the definition of *middle agents*. Note that these types of agents, which are described below, are defined so vaguely that sometimes it is difficult to make a clear differentiation between them.

- *Facilitators*: The agents in the system will surrender their autonomy to the facilitators in the exchange for the services they provide [18]. These facilitators will monitor requests received from their subordinated agents and try meet them, they would also coordinate the activities of the agent in the environment they run.

- *Mediators*: These agents are responsible for investigating and exploiting the encoded knowledge provided by a level of agents and to make it available by creating services for higher level applications [54].

- *Brokers*: The agents in the system will send requests to the brokers and the brokers will perform actions using a combination of their own services and ones provided by other agents [12].

- *Matchmakers and yellow pages*: These agents match up the demands from service requesters agents with services advertised by provider agents based on the skills [5] [12].

- *Blackboards*: The agents in the system will send requests to the blackboard agents for them to hold it and pass it to their agents for processing [39] [10].

3.5 Agent Communication Protocols

Since communication is a very fundamental concept for a system with more than one agent, there exist a set of languages and formats developed specifically for agent interaction. A number of these has directly been informed and influenced by speech act theories.

Here is a brief review of some of these agent communication formats and languages:

3.5.1 KIF (Knowledge Interchange Format)

This format was originally developed for expressing properties of a particular domain rather than a language in which the domain messages would be expressed [14]. Therefore, KIF is used to express the messages' *content*. KIF is closely based on first-order logic recast in a LISP-like notation. It provides for the representation of knowledge about the representation of knowledge and it provides for the definition of objects, functions, and relations. According to [56], for example, by using KIF it is possible to express:

- The properties of objects in a domain (e.g. "Mary is a dancer" - Mary has the property of being a dancer).
- The relationship between the objects in this domain (e.g. "Mary and John are siblings" - the relationship of being siblings exists between Mary and John).
- The General properties of a domain (e.g. "Every person has a father").

The objects in the case of a multi-agent system are the agents.

3.5.2 KQML (Knowledge Query and Manipulation Language)

According to [56], KQML is a message-based language for handling agent communications. This language defines a common format for messages. Object-oriented notion can be used for describing the KQML messages as an object. Each messages has a:

- *performative*: this can be described as the class of the message.
- *a number of parameters*: these are attributes/value pairs. These could be thought of as instance variables.

Although KQML was massively taken up by the community of multi-agent system developers, this language was also criticized on many levels mentioned below:

- since the basic KQML performative set were rather fluid, many different implementations of KQML were developed which were not interoperable.
- the mechanism for transporting messages were never well defined which meant that it was different for KQML-talking agents to communicate.
- the semantics of this languages were never rigorously defined. It became hard to determine whether the agents talking this language were doing it "properly".
- An important class of performative called *commisives* were missing which enables the agents to commit to each other. This property is important in most scenarios in which agents need to *coordinate* their actions with one another.
- The set of performative was overly large and rather *ad hoc*.

This led to the development of a new Agent Communication Language (ACL) by Foundation for Intelligent Physical Agents (FIPA) which was closely related to KQML but was also different to it on many levels.

3.5.3 FIPA ACL

FIPA began its work on developing standards for agent systems in 1995. The main outcome of this movement was the development of an agent communication language by FIPA in 1999.

This language is very similar to KQML as it defines an "outer" language for messages. FIPA ACL defines 20 performative for defining the intended interpretation of messages and it does not mandate any specific language for message content. However, the concrete syntax for FIPA ACL messages has a very close resemblance to the syntax for KQML.

According to [56], FIPA ACL is similar to KQML in terms of structure and the message attribute fields, however, the most important differences is in the collection of performative they provide. The other important difference is in the formal semantics. Given that one of the most frequent criticisms of KQML was the lack of adequate semantics, the developers of FIPA ACL decided to give comprehensive formal semantics to their language.

3.6 Multi-Agent Systems' Development Tools

Following from the 3.1 Section, the natural definition of multi-agent system would be a collection of one or more agents in a system. The agents can be of any of the forms described in 3.1 and will interact in one or many levels to achieve the desired goals.
A collection of these agents will provide us with the software framework needed for building a complex software system. This complex software system is composed of a set of interacting components, in other words agents, each of which is capable of performing a certain task. Such a system is completely modular and the input to an agent and the current internal state of it determines the future internal state of each component.
There exist many tools out there for build and design of multi-agent systems (abbreviated to MAS).
Here is a review of a few MAS development tools:

3.6.1 JADE

The Java Agent Development Framework is a software development environment for easy development of agent-oriented applications. This environment would ensure that the agents developed within this system will comply with the specifications FIPA has provided for multi-agent systems. Therefore, the main goal of JADE is to provide a simple development framework for MAS whilst enforcing the compliance with FIPA standards. This is done through a complete set of services and rules for agent development.

Consequently, as presented in [2], JADE offers the following development facilities:

- FIPA-compliant agent platform: This includes the Agent Management System, the Directory Facilitator, and the Agent Communication Channel. As soon as the agent platform starts up these three agents are automatically activated.

- Distributed agent platform: Provided that there is no firewall between the machines that JADE environment is running in, the agent platform can be run in a parallel manner across distributed platforms. JADE runs as one Java application which means one Java Virtual Machine will run on each machine. Each agent will be implemented as

one Java thread and communication between the agents running on the same machine is handled by Java events which makes them effective and lightweight. JADE schedules parallel tasks executed by each agent to be run in parallel.

- FIPA-compliant Directory Facilitators: These can be started at the run time to allow for implementation of multi-domain applications. The domain here is a logical one as is described in FIPA97 Part 1. More details on agent management specification in this version of FIPA can be found in the official homepage for FIPA (Foundation for Physical Intelligent Agents) in [16].
- Programming interface: This would simplify the way the agents' services would register with more than one domain.
- Transport mechanism and interface: This would allow agents to send messages to other agents and receive ones from them.
- FIPA97-compliant IIOP protocol ([16]: This means that one could connect different agent platforms.
- Light-weight transport of messages in the same agent platform: In order to avoid marshalling and un-marshalling procedures the messages are transferred as Java objects as opposed to strings, If the sender agent or receiver are not running on the same machine, the message is converted to/from FIPA compliant string. This is done automatically which means that the software developer would only need to handle the same class of Java object since the conversion is done behind the scenes and handled by the JADE environment.
- There is a library of FIPA protocols for interactions available to use by developers.
- The agents are automatically registered with the Agent Management System upon their start-up.
- FIPA-compliant naming service: The agents are provided with a Globally Unique Identifier by the platform upon their start-up.
- Graphical user interface is provided for managing many other agents and machines using the same agent as starting point. The interactions and agent activities occurring on various machines agents are running on can be logged and monitored.

3.6.2 FIPA-OS

According to [7] FIPA-OS is a toolkit allowing for fast development of agents which are FIPA compliant. This toolkit includes most of FIPA experimental specifications. It also has the benefit of being managed as an Open Source project which means that developers around the world will continuously carry on improving this system. This makes this system a very good choice for developers. For more information see [7].

3.6.3 DECAF

As discussed in [23] DECAF (Distributed, Environment-centred Agent Framework) is a toolkit which supports a strong and well-structured software engineering approach for development of multi-agent software systems. This toolkit provides the developer of complex software systems with a stable environment to easily be able to design agents and then to rapidly build and run them. The services provided by DECAF cover all architectural aspects of developing agents in terms of communication , planning

DECAF provides the necessary architectural services for developing intelligent agent on a coarse grain in terms of communication, planning, scheduling, execution monitoring, co-ordination and even down to learning and self-diagnosis, too. This in a sense becomes the internal "operating system" or an agent. And on the surface the developers would have very limited access to the internal part of the system to modify aspects mentioned above such as communications.
As mentioned above one of the major goals behind DECAF was providing the developers with rapid agent developments. This is achieved by the lack of need for developers to worry about programming some of the more internal components such as the agent interactions which can be very tedious and time consuming. The developer which uses DECAF would not have to write any components to handle agent communications and also to handle multiple invocations of same agents. He/she would also not need to write further components for parsing any incoming messages or scheduling them for delivery. There is also no need for programmer to learn about any specific API (Application Programmer Interface) in order to use DECAF.
The DECAF running on basic mode is comprised of three main components:

- Agent Name Server (ANS): This simply acts very similarly to a Domain Name Server or the port mapper on UNIX operating system. Upon the start up of a new agent, this agent will need to register with the ANS. Therefore, this ANS acts a post office or the hub of the system which knows about all existing agents in the environment and can be consulted for finding agents.

- Agent Program or Plan File: This is the programming of the agents and it is produced as an output from the Plan Editor. Each agent is made of several capabilities and potential goals. It also holds a collection of actions which could either be planned or run to reach its assigned goals.

- The DECAF system framework: This is the main DECAF framework in which the two above main components exist.

3.6.4 ZEUS

As discussed in [25], this is a framework created as part of the Midas and Agentcities research projects at British Telecom in late 1999's and early 2000's. The main motive behind ZEUS as discussed in was to provided a generic, easily modifiable to suit individual needs, collaborative and at the same time scalable toolkit for developing software MAS. ZEUS is a collection of classes developed in Java which instantly makes the developed agents portable. This means they will be able to run on multiple hardware across various platforms. These classes which form Zeus are divided into three main groups which are library of agent components, agent construction tool and agent visualisation tool. The component library is naturally comprised

of a series of Java classes for building the internal parts of each agent. The collection of these classes is used for implementing the functionality of a collaborative agent on a fine scale.

According to [25] the important part of this toolkit includes:

- The tool for building agents provides the developer with a series of editors which are based on ZEUS agent development methodology. This tool has got a very user-friendly interface in the way that it allows the developer to interactively and visually design and create agents by adding attributes to them.

- A visualisation tool which is available for developers to view a society or collection of ZEUS agents and analyse or edit them. It is important to note however that this visualisation agent is simply a ZEUS agent itself and it is built using the same components as other agents in ZEUS societies. To allow developers to closely monitor the collective behaviour in the agent society, this visualisation tool constantly queries all existing agents in the society for their current state and the running processes which then helps it to build a real-time model of the society at any given time. Once the model is available, the different aspects of the agent societies can be viewed using five different visualisation tools.

3.7 Conclusion

There are a few reasons why at this stage it was decided to develop the agent framework as a custom framework rather than using one of the off-the-shelf agent building toolkits mentioned in previous sections. These reasons have been highlighted below:

- Only one of the above toolkits is an agent development kit specifically designed for building multi-agent complex systems. This is DECAF. The reason it is important to use a tool which has been specifically designed to solve complex systems is so that the emphasis would be on easy modularization and expandability rather than sophisticated communication between agents or the level of autonomy.

- The problem with most of these systems is their lack of support for Python. Python is a scripting programming language with facilities for calls to functions which are written in other high-level languages such as C. The reason Python is preferred as a programming language in this work is because languages such as C or Matlab can be incorporated within Python. Also, Python is a multi-platform language which means that the Python agents' algorithms will run in more than just the Windows operating system without the need for a rewrite of the framework. Java also benefits from being multi-platform. JADE and ZEUS are implemented in the Java language. However, Java is an interpreted language as opposed to C which is a compiled one. This makes Java an order of magnitude slower than C despite the advanced use of compilers. Python is also interpreted but it has better support for passing parameters to pre-compiled functions, which makes it a more preferred choice. Therefore, if Python is used with C, the pre-compiled functions in C which would receive the parameters from Python would run faster than if Java was used. Other reasons why Python is a preferred choice are discussed later in Section 6.3.

- In most of the existing tools the messages passed between the agents are not simple ones. These systems are very suitable for building MAS which require the agents to

cooperate, *coordinate* and *negotiate*. In terms of this project, this is not needed. The only communications which take place between agents are very simple ones. These are mainly for alerting the router of a new agent, terminating an agent, or what agent to call next and with what data. No communications take place directly between agents.

- Some of the systems above use notions such as ontology. The ontology is needed if the agents have to deal with more than one domain and the ontology is used to describe the properties of a domain and the relationships between the things in this domain. At this stage, there is no need for including this in the project since the agents in the MAS required for this project do not need to have a sophisticated concept of the environment they are running in. However, this is an extension that could be made if the requirement arises for the use of agents which are environment-aware.

- Due to the nature of the problem, the only properties the agents need to possess are social ability and reactiveness. This would result in the need for a very lightweight MAS framework with non-automated agents. Most of the existing MAS development tools are suited to MAS, whose agents have more than the mentioned properties. Therefore, their methodologies will add overhead on the design of the framework needed in this work.

- The main emphasis here is the "multi" aspect of the MAS rather than pro-activity or automation.

- One of the main issues in the use of MAS in this work is a quick implementation due to the need for quick access to the results required for the completion of gait recognition program. Therefore, the high level of robustness encouraged by off the shelf systems is less of a priority.

Therefore, the decision was made to develop a light-weight agent architecture from scratch. This is very simple framework comprised of several agents for running gait recognition algorithms and a router. The type of agents is the closest to Wooldridge-Jennings agent as mentioned in Section 3.1.1. However, out of all four properties mentioned for Wooldridge-Jennings agents, the ones in this work possess the *social ability* and *reactivity* properties.
The communication protocol which will be used is defined with a simple set of performative inspired by existing communication protocol such as FIPA ACL. (See Section 3.5.3) However, the set of performative is very small compared to FIPA ACL. This is because most performative types defined by FIPA ACL are not needed by this work due to the simplicity of communication between the agents. One example is that the agents in this work do not need to coordinate their actions, this means that a whole set of performative called commisives will not be needed.

The other reason for not strictly using FIPA ACL is the lack of a need to include formal semantics in the communication language required. The communications in this work are very simplistic and only occur between each agent and the router.

However, further investigation was made into all the above off-the-shelf development tools, design methodologies and communication languages for inspiration and also for learning about the strategies used for solving issues common to multi-agent systems.

Chapter 4

The Design of a Lightweight Multi-Agent System

As discussed in the previous chapter, the idea for this project is to develop a lightweight multi-agent system which is inspired by existing multi-agent systems, but is a lot simpler. As seen in Chapter 2, the reason a multi-agent system will be used is that this paradigm allows for distributed processing, which can increase the speed of running gait identification algorithms. It encourages a modular design which yields greater flexibility and maintainability for an application of a research nature, in which new techniques may be developed regularly. Finally, distributed resources are also supported, which means that the data can be processed remotely without needing to be transferred to the local machine. The router-agent concept in this framework would also create a central place at which failures can be monitored.

Therefore, the aim is to develop a framework which is expandable and easily maintainable. Such a framework allows a group of researchers to carry out the development of a set of algorithms and to interface them for maximum inter-operability.

It also enables the users to run their algorithms in a distributed manner on a cluster of machines for cutting down the total processing time. In the rest of this chapter, the design issues for this lightweight multi-agent system are discussed.

4.1 A Simple Multi-Agent System Architecture

In this section, we will look at a generic multi-agents system, exploring the responsibilities of its components and the ways they communicate with each other.

A lightweight multi-agent system will be comprised of a *Router* agent and an arbitrary number of *standard* agents. This is illustrated in Figure 4.1.

4.1.1 Router Agent

The central part of this multi-agent system is the router. This is a vital part to the system since it acts as a "post office". The location of the router is known to all of the agents in the system. Therefore, at the system start-up, all agents subscribe to the router and from this point the router passes messages from one agent to another. The router keeps a table of all existing agents in the system including their addresses and status. When an agent dies or is stopped, the router will delete the information about that agent from its look-up table.

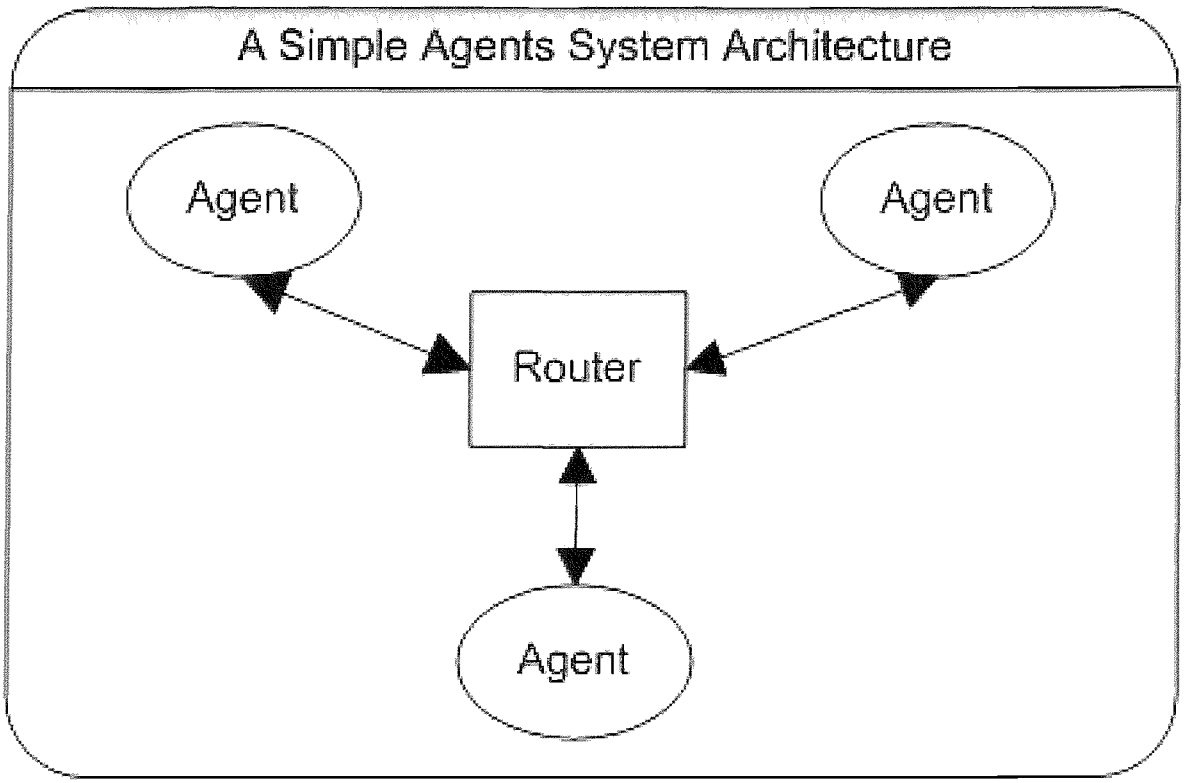

Figure 4.1: Components of a simple multi-agent system and their communication

In some systems, the router acts as the central post office agent, however, it is not responsible for passing messages between agents. This is due to the fact that the message passing could create a bottle-neck and slow the system down. In this work, the router acts as a message passing centre too and this is due to two main reasons. One reason is that the overall framework implementation will be a lot simpler, as everything will be kept inside the router. There will not be a second agent for message passing because such an agent would need to have constant access to the router's agent look-up table. The second reason is that, in this example, the ratio of the number of agents to the time they take to complete their work is quite low, hence the issue of creating a bottle-neck at the router side due to a vast amount of message passing is almost non-existent. Therefore, all message passing is done at the router.

The router in this system is very similar to a previously mentioned yellow page server, however its job is very simple in that it only implements some simple communications functionalities needed for agent-to-agent interaction. A yellow page server, on the other hand, would need to perform the job of matching advertised needs to advertised requests posted by the system's agents. In this system, the matching of requests to availability is done by a separate agent.

The messages are very simple and contain three parameters which are "performative", "target" and "sender". The "sender" and "target" are respectively the identification of the agent who sent the message and the agent who is meant to receive the message. The "performative" is a description of the message. For example, a typical "subscribe" message which would arrive at the router will have the "performative" set to "subscribe", the "target" set to

the "router" and the "sender" set to the unique ID of the agent which is trying to subscribe.

The Agent Communication Language (ACL) used in this work is created specific to the needs of this work but it is highly inspired by FIPA ACL which was looked at in the Chapter 3. In a simple manner, the life cycle of the system is that the agents starts up, they first subscribe to the router. They then wait for incoming messages to act upon and are killed at the end of their lifetime when their purpose is no longer needed. There is of course one more issue with the current version of this system and that is the consequences of the router dying.

In the current version of the system, if the router dies, all agents will lose contact with the rest of the system and messages can no longer be passed on. There is a logging system which keeps a list of all connected agents, however, there have been some thoughts on how to improve this. One way would be by using a broadcasting facility provided by the operating system. Just before the router dies, it broadcasts the latest status of its agent look-up table and a secondary router will be started to take over. As result of this broadcasting, this second router will have access to the latest status of the system and will restore an agent look-up table based on the most up-to-date information as logged and broadcast by the previous router.

4.1.2 Standard Agent

A standard agent is any user agent in the multi-agent system which is not the router. Every standard agent can send messages to other agents in the system via the router. Each agent has a unique postal address that only the router knows about.

As soon as an agent is started and initialised, a *subscribe* message is sent to the router which includes its name and address. This information is stored in a table by the router. From this point onwards, the agent will be able to *receive* and *send* messages from and to other agents in the environment via the router. Occasionally it will also need to talk directly to the router to inform it of any important changes to its state. The most important of which is when the agent finally exits the system.
A full description of the above components in our multi-agent system can be found in the following section.

4.2 The Design of a Lightweight Multi-agent System Architecture

This section looks at the design of a generic lightweight multi-agent system.

4.2.1 Base Agent or Worker

The first and simplest component to look at in the design of this framework is an *agent.* The base *agent* in this system is a component which has got three main layers. These are:

- *Communication*
- *Message parsing*
- *Functionality*

Communication layer

Referring to the simple agent architecture described in 4.1 these agents start up in the typical agent environment and start to communicate with each other via the router. This layer establishes and maintains the agent's communication with other agents via the router. It is also responsible for *sending* the messages and *listening* for the incoming ones.

Message parsing layer

The components in the agents system communicate by exchanging messages with each other. Each message is comprised of a number of fields which contain information such as the message sender, target and the action that needs to be taken.

Once a message has been received by the receiver part of the agent's communication layer it gets passed to the message parsing layer. This then interprets the content of the message and carries the actions that need to be performed.

Functionality layer

There are two types of functionality present within an agent. The general "house-keeping" functionality and the "work" functionality.
The "house-keeping" functionality contains standard actions that all agents regardless of their type need to perform. An example is to receives a "disconnect" message from the router. The "work" functionality is the algorithm part which is specific to each agent. This part is not implemented at base agent, but in a separate part called the "engine".
By using a separate "engine" part, a layer of abstraction is added to the framework design in order to simplify the addition of new algorithm agents for end users. It also helps to keep the agent concept as generic as possible, as all the specific algorithms functionalities can be contained wholly within the engine allowing the abstract agent to be used for developing different types of agents.

4.2.2 Router

The router component is a special agent which simply acts like a "post office", and as such most of its communication layer is inherited from the base agent class or are implemented in a similar way. The location of the router agent is fixed and known to other components including the agents in the system.

Upon the start-up of the system, the router is launched and the rest of the agents subscribe to the router as soon as launched by the user.

The router keeps a table of the active agents in the system which contains their names, their types and their locations. Everytime an agent subscribes to the router, a record is added to this table. The router then binds the agent to a unique port number which is randomly generated by the system. This port number combined with the IP address of the system the agent is running on, forms the unique postal address in the router's look-up table.

Using port numbers as part of the agents' address could of course introduce a limitation. This would mean that the maximum number of agents on each machine will be limited to the number of existing ports. However, this number applies to each machine. And, knowing that our system would only require to run a handful of agents on each machine before the efficiency of running a multi-agent system is lost, this is not a real limitation. Since the unique ID is

comprised of the IP address of the machine as well as the port number, other unique ID's using the same port number but different IP addresses can be constructed if the agents are running on other machines.

The main responsibility of the router is to deliver the messages from one agent to another. Therefore, all the messages in the system are sent to the router. There are two major types of messages. One type is the messages that are addressed to the router (e.g. Subscribe, Kill) and others are the ones to be forwarded onto the other agents. The messages will not be forwarded on unless the target agent has already subscribed to the router.

Whenever the router receives a message which needs to be forwarded to an agent in the system, it performs a search on its agent look-up table to find the address of the destination agents (their IP address and port number). The message can then be forwarded to the target agent using this information. This system is highly dynamic since agents can be started up at any time and as soon as they subscribe to the router they become instantly available and can be contacted by other agents in the environment.

4.2.3 Advantages

The main question here is why use a multi-agent system whilst it is possible to develop the application using other frameworks such as distributed systems. Here is a list of some key characteristics that are specific to the multi-agent system framework which our system can directly benefit from:

- *Expandability*: This framework is highly compatible with new changes made and provides an easy plug-and-play interface for new agents to be added to the system, in a way that they are instantly identified and start to interact with the rest of the agents. Therefore, there is plenty of room for the system to grow without a need for a major re-write of the underlying system structure.

- *Modularity*: The system allows for each part of the application to be implemented in isolation without any concerns about how the other parts of system work. Therefore, as long as the programmer is aware of communication protocols (what messages to send and what messages is likely to receive) then they can write an agent which can interact with the rest of the system.

- *Parallel Processing*: The multi-agent architecture is a naturally distributed paradigm which can make a highly efficient use of a cluster of computers in which tasks (such as image processing or biometric algorithms) can run in parallel and interact as and when they achieve some intermediate results.

- *Maintainability*: This design allows for rapid maintenance such that, once written, underlying structure takes care of the low level hard to write parts of the system whilst several programmers can be working in parallel on adding new algorithms to the application and maintaining old algorithms.

4.2.4 Disadvantages

As with any architecture, for the benefits that are gained from this design there would be drawbacks to consider too. Some of these disadvantages are:

- *Dependence on bandwidth*: The performance of this multi-agent system is partly dependent on the available bandwidth of the networked cluster of machines that it is running on. This is because the more limited the bandwidth, the speed of the interactions will drop and therefore the overall processing time will increase too. The other dependency on the bandwidth is the file transfer from the main data source to where the agents are running. In a distributed system where one machine acts as a data store and others as data processors, where there is need for large files to be transferred from the data store to the data processors, a large bandwidth is needed. Otherwise, the transfer process will be slow and the efficiency gained by parallel and distributed processing by using a multi-agent system will become questionable. However, in our example, the cluster benefits from a transparent, shared file system. The files are not that big and the network is reliable. Therefore, the bandwidth limitation for this example is not a major concern.

- *Overheads*: The communications between the agents and the router in this multi-agent system can introduce some overheads on the bandwidth and slow down the overall performance. This may or may not be tolerable depending on the ratio of time the agents spend on processing their tasks to the time they spend on communications. If this ratio is high, then the overall processing time will not be affected as much and the benefits of using this system is justified. In our example application, the tasks of the agents are expected to be time consuming, thus make the overheads associated with the agents communications of little significance.

- *Data Reliability*: One of the main issues with this design is that the data transport and integrity needs to be carefully designed and implemented, otherwise the reliability of the system can be questionable. This is due to the fact that the transfer of the data between the individual components of the system is one of the most sensitive parts of this architecture. This again is not of great significance for this work, since our system is not a safety critical one. This means the loss of data would only result in a task not being done which can be detected later by inspecting the Slavedriver task logs. Hence, this is not a critical issue and things such as health and safety are not the primary functions of this example system, the lack of reliability can be tolerated to a reasonable amount.

- *Security*: A multi-agent system can potentially be an open system, since the agents can run anywhere on any machine and as long as there is a network, they can communicate and carry out their tasks. As with any open system, the security measures need to be taken into account very seriously, otherwise potential problems may occur. In this example, the agents will be very limited to where they will be running, which is a local cluster of machines behind a firewall, therefore, security is again of little significance.

Chapter 5

Gait Extraction

So far, it has been explained that the problem to solve in this work is to propose and develop an efficient framework for the construction of the gait identification system. In the last few chapters it was explained that the gait identification system is complex in terms of multiple components and their interaction. Therefore, the proposed framework is a generic lightweight multi-agent system that the gait identification algorithms will be developed within.

This chapter takes a more in depth look at gait recognition and biometrics to give a better insight into the nature of the challenges involved in building such a system. This will demonstrate the benefits that are gained from developing this work using a multi-agent system framework.

A brief summary is given about gait identification as a biometric for human identification. Later there are sections on gait identification algorithms researched in Southampton University and the image processing algorithms and techniques which are used in these algorithms.

5.1 Gait Identification

Gait is an emergent biometric aimed essentially to recognise people by the way they walk. In many applications of person identification, established biometrics can be obscured. The face may be hidden or at low resolution; the palm is obscured; the ears cannot be seen.

However, people need to walk, so their gait is usually apparent. This motivates using gait as a biometric and it has recently attracted interest. Gait is attractive since it requires no subject contact, in common with automatic face recognition.

Apart from being perceptible, another attraction of using gait is that, motion can be hard to disguise. Consider for example a robbery: The robber will need to make access either quickly, to minimise likelihood of capture, or without being obvious in order not to provoke attention or to move quickly.

Clearly, there are limits to the use of gait as a biometric, a detailed study of the limitations is expected for the development of a technique. However, it is likely that footwear can affect gait, as can clothing. Equally, physical condition can affect gait such as pregnancy, affliction of the legs or feet, or even drunkenness [29].

However, some or all of the above negative factors apply to other biometrics too such as face, ears or fingerprint. The view that gait can be used to recognise individual is not new for example Shakespeare has used a number of adjectives to describe ones gait. Some example are princely, lion's, heavy, humble, weary, forced, gentle, swimming and majestic.

Essentially, we use computer vision techniques to derive a gait signature from a sequence of images. The majority of current approaches analyse an image sequence to derive motion characteristics that are then used for recognition. Early results by these studies confirm that there is rich potential in gait for recognition.

In the following sections, a few gait identification algorithms which are researched in Southampton is reviewed. These are as follows:

- Symmetry [27]
- Velocity Moments [45] [51]
- Standard Average Silhouette (SAS) [53]

Before we could apply these algorithms to the subject's video which is in digitised colour video format, some pre-processing would need to be done. In this case, we have two main pre-processing stages:

- Generating Silhouettes: To convert RGB digitised video into a sequence of silhouettes. These silhouettes are binary images of white subject on a black background.
- Period Detection: To determine the start and end frame of a full period of walking cycle. This is used by gait algorithms to produce gait signatures from a subjects walk cycle.

The following sections look more closely at each of the pre-processing stages mentioned above.

5.2 Generating Silhouettes

To produce binary subject silhouettes from a video several steps are performed. These steps are described briefly below and illustrated in Figure 5.1 when applied to a subject's digitised video:

- Calibration: The video is calibrated to correct the radial distortion errors which causes straight lines to appear at an angle.
- Chromakey: This step will mark some known background pixels to assist with extracting the subject . The data has been recorded in a lab with a fixed background cloth which is green and the track that the subjects walk has also been painted a very dark shade of green. Therefore, with two passes of chromakey most of the background will be eliminated.
- Cropping: The chromakeyed frames from the video are then cropped with fixed values adjusted to the gait lab parameters. This would crop the scene to an area of track which apart from the walking subject does not include unnecessary objects light stands and tripods in it.
- Connected Component: This algorithm will find the biggest connected blob in the given binary image. The local neighbourhood algorithm used for this purpose is called the *blob colouring algorithm* described in [1].

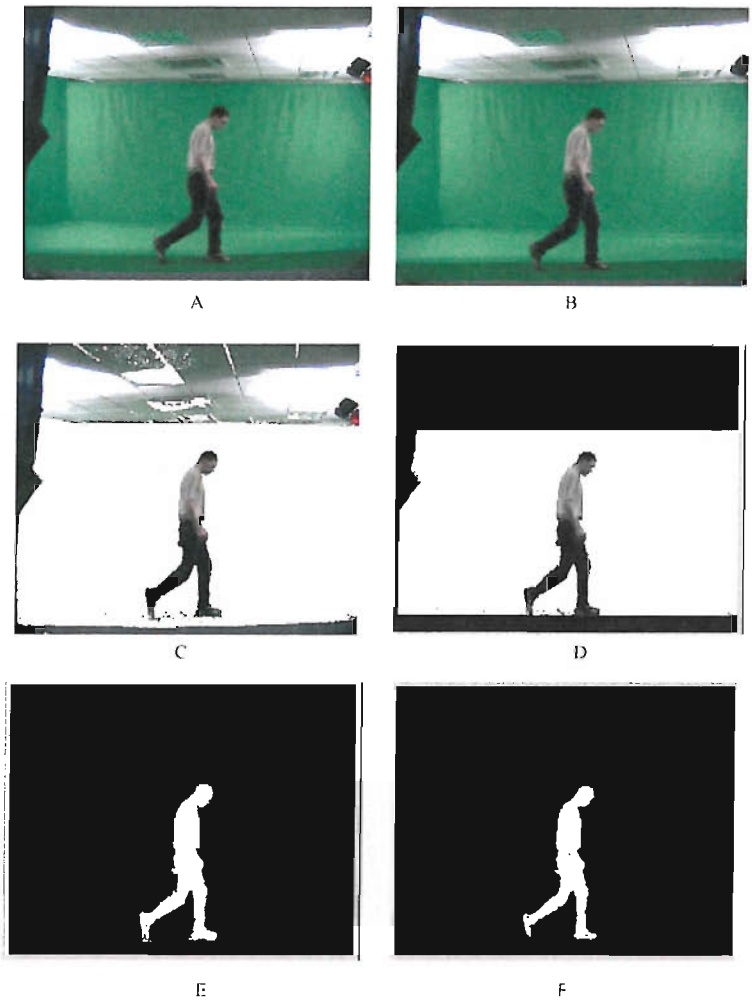

Figure 5.1: The results from various pre-processing steps in generating silhouettes from a subject's digital video (DV): A) Raw DV frame B) Calibration C) Chromakey D) Cropping E) Connected Component F) Three times Erosion and three times Dilation

As reported in [42], this algorithm has two passes. In the first pass, colours are assigned to image pixels by using a three-pixel L-shaped mask. Colour equivalences are established and stored, when needed. In the second pass, the pixels of each connected region are labelled with a unique colour by using the colour equivalences obtained in the first pass.

After the frame has been processed through previous stages mentioned, the biggest connected blob in the scene will definitely be the subject.

- Erosion, Dilation: According to [42], these two processes will assist with cleaning the silhouettes (e.g. filling the holes, smoothing the sharp edges, etc.). *Erosion* is a shrinking operator which shortens the object's limbs and expands its holes. Whilst, *Dilation* is an expanding operator and tends to shrink object holes and to expand object's limbs.

The output from the above stages is then a series of binary images with the subject in white on a black background. This is the input to the next stage, which is the Period

Detention.

5.3 Period Detection

As mentioned before, this process determines the start and end of a full gait cycle in the given sequence of a walking subject's silhouettes. This stage is needed since the current gait recognition algorithms developed in Southampton rely on calculating the gait signature over a complete walking cycle, which in most cases starts from when the subject's heel hits the floor (heel-strike) until the same heel touches the floor again. This usually defines a walking period of 25 to 35 frames.

The way this process determines the beginning and end of a walking period is by plotting the distance between the subjects legs throughout the duration of their walk. The period is then determined from the (n)th leg distance maxima to the (n+2)th one. However, the period could also be determined using alternating minima.

In each given video sequence, the process is likely to find more than one gait cycle. However, only the most central one is marked to be used for gait feature extraction.

Now that the pre-processing stages have been explained in the above sections, some gait feature extraction algorithms can be looked at in more details. As mentioned before these are: *Symmetry*, *Velocity Moments* and *Standard Average Silhouette(SAS)*.

5.4 Symmetry

This method for automatic gait recognition is simply based on analysing human motion based on its symmetry.

It has been described in [27] that this technique stems from the psychologists point of view which classifies human gait as a *symmetrical* pattern of motion.

Symmetry is a fundamental (geometric) property suggesting it to be an important principle of perception [27]. An object is said be to symmetric if its shape remains unchanged by the application of symmetry operations. Boolean symmetry operations can only assess symmetry when the shape of the object is known in advance, rendering them inappropriate in most cases. The discrete symmetry operator can estimate symmetricity without the knowledge of the object's shape, unlike feature extraction operators that find a shape by relying on its border or on general appearance.

The symmetry transform assigns a symmetry measure to each point in the image and is determined with respect to a given point-symmetry group. In [46], Reisfeld reported that the performance of the symmetry transform is not affected by existence of several objects in the scene. This depends on values selected for controlling parameters.

The results in [27] show that gait can indeed be recognised by analysing its symmetry.

5.4.1 The Extraction of Symmetry

In [27] it has been explained that the discrete symmetry operator uses edge maps of images from the sequences of subject silhouettes to assign symmetry magnitude and orientation to image points, accumulated at the midpoint of each analysed pair of points.

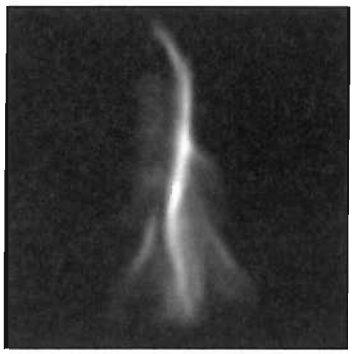

Figure 5.2: Symmetry signature map

The symmetry operator is then applied to give the symmetry map. For each image sequence, the gait signature, GS, is obtained by averaging all the symmetry maps, that is:
$GS = (\sum_{(j=1)}^{N} S_j)/N$
where N is the number of symmetry maps in a sequence. See [27].
An example of an average symmetry map can be seen in Figure 5.2.

5.5 Velocity Moments

This technique as described in [51] is developed based on the idea that the motion of a moving object in a sequence of images can be described and modelled using Zernike velocity moments. These moments are an extension to the orthogonal Zernike set of moments in the way they include velocity information about the moving objects through centralised moments. The early experiments and analysis proves that these moments perform very well used for analysing human gait sequences. This shows an early estimation of a good recognition rate and the gait signature comprised of these moments appears to be small and compact.

In the early sixties, Hu [28] was the first person to apply classical moments to two dimensional images. The images Hu used were noise-free and he used a series of hand-written characters to test the validity of his technique. He then performed further experiments which resulted in showing the performance of traditional moments dropping in the presence of noise or in cases where the object was not very visible.

The new method proposed in [51] is aimed at producing more compact description of motion of objects in a sequence of images. This description is also aimed to be less correlated. There is also the added advantage that the result description is invariant between and within the image sequence to both translation and scale. This makes this technique ideal for a real life scenario in which camera zooming exists in the image sequence across frames or in the same frame.

This method is an extension to the well established and well proven Zernike moment set in pattern recognition [45], which is usually used for image sequences with noise.

As discussed in [51], the main basis for this new approach is a standard procedure for describing a region. This procedure is simply used in a generic way for describing the moving object by holistic measures. The first step was to calculate velocity moments up to order of 12 for all sequences of spatial templates in gait database. Then the new moments are calculated from one heel strike to next to form a complete gait cycle (this is usually equivalent to two

steps). Once calculated, the magnitude of the moments are then studied to reduce the size of the selection problem. For more information on this technique please see [51]

5.6 Standard Average Silhouette (SAS)

The standard average silhouette is simply an average of silhouettes over a collection of silhouettes in one full gait cycle. Using the period data for a complete gait cycle and the subject's silhouette data, this process outputs a portable pixmap (ppm) image which is a signature map for a certain subject in the given gait cycle.

This method first locates the subject in the binary frame using a function that returns the position of a bounding box framing a non-black area in the binary image which in this case should be the subject. Then, the centroid of this white blob (subject) is calculated.

Using the centroid data and bounding box dimensions, the maximum distance from the centroid to the bounding box sides is found and a square can be defined to crop out of the image which includes the subject on a centre of a smaller area of background pixels. This is then resized to a 64 x 64 image.

These 64 x 64 subject silhouettes are then added together over the total of frames that form the complete gait cycle of a given subject in a given sequence. The output is a ppm image file of accumulated binary silhouettes.

As described in [53], it is shown that the average silhouette includes a static component of gait (head and body) as the most important image part, while dynamic component of gait (swings of legs and arms) is ignored as the least important information. At the same time ignoring dynamic part of gait can result in loss in recognition rate in some cases, and the importance of better motion estimation is under-lined. An example SAS map can be seen in Figure 5.3

Figure 5.3: Standard Average Silhouette (SAS) signature map

5.7 Conclusion

The above was an overview of various image processing algorithms involved in extracting gait and processing it for identification of subjects. The gait extraction processes transform raw digitised video to gait signature maps.

The diagram in Figure 5.4 shows where the above algorithms are situated in a typical processing flow for retrieving gait signature from a subject's walking input data.

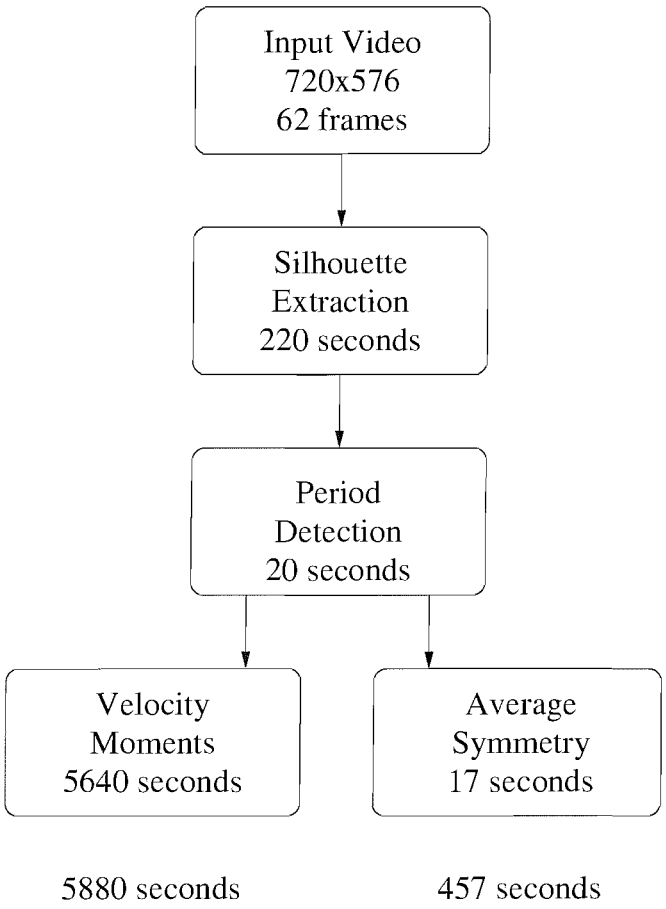

Figure 5.4: A typical flow of processes to produce gait signature from raw digital video files using two different gait algorithms: average symmetry (see 5.4) and velocity moments (see 5.5)

In the next chapter, the development of the lightweight multi-agent system and the ways in which the gait algorithms above will be embedded into the proposed framework of Chapter 4 will be discussed.

Chapter 6

Multi-Agent System Gait Applications

In the last chapter we looked at gait identification as a biometrics science and some of the algorithms involved. Now we are going to look at how the two main gait identification applications discussed in 1.2 are developed inside the multi-agent system framework proposed in Chapter 4.

As mentioned in 1.2.1, there are two main possible scenarios for gait recognition applications that are of importance to this work. One is the *Multi-Agent Gait Recognition Demonstrator (MAGRD)* and the other one is *Multi-Agent Gait Recognition Batch Processor (MAGRB)*.

6.1 Multi-Agent Gait Recognition Demonstrator (MAGRD)

This application was not fully implemented due to the high demand within the group for the development of the second application which is the batch video processor.

As mentioned in Section 1.2.1, this application is comprised of a video browser agent and a number of gait recognition and image processing agents. The user can open a video file, mark up a section of interest and then apply one or more image processing and gait recognition processing techniques to this video file. The result is reported back in a meaningful manner for further off-line analysis.

The idea is that, having implemented this scenario using the agent framework, the system's menus are created and updated on-the-fly from a list of available agents in the system. Some of the components for this application such as the video browser and mark-up system have already been implemented. However, there are some fundamental questions that still remain un-answered. These would have been discussed and answered in the event of full implementation of this application.

Some of these are as follows:

- How does the browser agent obtain information about the functionality of existing agents in the system?
- How do existing agents in the system communicate with the browser agent?

The agent framework provides the full support and implementation of message passing between the agents, however, no protocol exists to allow the browser agent in this scenario to

obtain necessary information and communicate to the rest of the agents in this environment. This demonstrator initially runs on one machine but it can be easily extended by distributing the algorithm agents on various other machines and calling them as and when needed to process data. This will help improve the overall processing time of the system given a set of tasks.

As a different approach to a demonstrator, another scenario was implemented for a real-time automatic gait recognition system which is comprised of the following components:

6.1.1 Video Capture Agent

This agent receives live video as input from a digital camera and performs an auto-detection of when the person enters and leaves the scene. The video data begins with the entering subject and ends with the leaving subject. This is then saved and the filename is sent to a processing agent.

6.1.2 Processing Agent

This agent holds a list of processes to run on the data in order to extract the gait signature from the input data for the purpose of subject recognition. Therefore, the image processing (e.g. silhouette generation in 5.2, period detection in 5.3) and gait algorithm agents (e.g. symmetry in 5.4) are called one after another until the gait signature is retrieved and saved. The name of the signature file is then sent to the gait demonstrator agent for enrollment, viewing or verification.

6.1.3 Gait Demonstrator Agent

This agent has a user interface which runs in either *Enrollment* or *Recognition* mode. It also contains a video browser agent. The following steps are taken whilst the gait demonstrator agent is running:

- Once a gait signature has been extracted from a captured input video, the gait demo agent is called with the name of the raw video and the gait signature image.
- If in Enrollment mode, the input video is viewed in the browser with the signature map on one side of it. The user can then enter the subject's id in the provided box and hit "accept" to add an entry for the subject in the gait recognition database.
- If in Recognition mode, the input video is again viewed along with the signature map, however, the user is provided with a "score" button which allows them to see how the newly captured subject's video scores when matched to the rest of the database. In the ideal case, the first few top score entries in the list would belong to other video sequences of the same subject.

This system was demonstrated successfully in an inaugural lecture on Gait Recognition as a Biometric within the ISIS group.

The rest of this chapter concentrates upon the implementation of the second application which is the Multi-Agent Gait Recognition Batch Processor (MAGRB).

6.2 Multi-Agent Gait Recognition Batch Processor (MAGRB)

As mentioned before in 1.2, the MAGRB system is an example of a real world application of multi-Agent system technology to solve a computationally intensive task in a *fast*, *scalable* and *modular* fashion.

The main challenge of the MAGRB application lays in finding an optimal solution for fast processing of large volumes of data. This is going to be solved by using the proposed multi-agent framework on a cluster of machines.

There is a central agent which knows about the jobs to be processed and it distributes them amongst the worker agents. Each agent will run specific image processing or gait recognition algorithm on a video file at any given time. This continues until no more jobs are left on the broker agent's list.

The implementation details of this application are divided into two main areas:

- Core Multi-agent system framework
- MAGRB's specific agents

However, before we look at the two above areas, the decisions first need to be made about the *programming language* and the *communication protocols* to be used in the development.

6.3 Programming Language for Implementation

To choose the right language for implementing this framework is one of the keys to the success of the development. In order to choose a language the following criteria had to be met:

- Platform Independent
- Easily Maintainable
- Multi-Threaded
- Networking Support
- Interface to other languages
- To match the current user groups' language skill set

A short list of modern languages that could potentially meet these criteria was therefore drawn up and this list is as follows:

- C
- Java
- Python

6.3.1 C

C is possibly the most popular modern programming language and there is a wealth of libraries already available for networking and other required functionalities. C is also the "fastest" of the languages being considered which makes it a good choice for the computationally intensive parts of the system. It is, however, one of the hardest to implement with, because it adds a few programming overheads such as memory allocation, handling, etc. This is due to its relative low-level simplicity. There are no real built-in runtime platform-independent features such as pathnames, etc. Also C is a compiled language which means porting work is required to enable the code to work on different platforms. At the same time, almost all computer systems come with C compiler where as Java and especially Python are a lot less common.

6.3.2 Java

Java is somewhat simpler to develop with than C in that it handles the memory allocation and destruction automatically (garbage collection facilities). However, there are many libraries to use in Java and this could be an unwelcoming characteristic to new developers. Java is better at handling the platform independence than C. This is one of the key principles behind creating Java. It also has a strong, powerful, yet simple network API, which makes the communications easier to program. However, Java runs on a virtual machine and therefore is somewhat slower in execution than C.

6.3.3 Python

Being a scripting language Python is inherently platform independent, and it takes this further with a number of modules for dealing with file access in a generic platform independent way. Python is the language most used by the current research user group and is therefore the best choice from a maintainability standpoint. The research group is generally skilled with using Python for development, therefore, this allows for easier future extensions to this work. Python's communications models are not as sophisticated as Java's ones, however, they are simpler to use than the C sockets libraries.

6.3.4 Conclusion

All of the languages short listed met most of the requirements.
However, the concluding points are:

- There were too many programming overheads in using C, too few features readily available, and too many simple functions available elsewhere to warrant the time spent in coding them in C. However, C does outperform Python and Java, both of which are slower languages, and as such is well suited for implementing the core algorithms which perform the data processing tasks. Java and Python are simply too slow to be used for this purpose.
- Python is an evolving language, and has an easy to use C interface layer and enjoys a strong following in the user group, so it is more desirable than Java for the development of the proposed multi-agent system framework.

- Python is slow in processing mathematical operations, however, the readily available C libraries exist which include some of these operations. One example is Numerical Python module which enables the programmer to perform time-critical calculations using the compiled C rather than Python. For more information see the official homepage of the developers of Numerical Python in [40].
- The proposed lightweight multi-agent system is not strictly a time-critical one, however, the reliability in programming the communications for it should be of a major significance.

Based on the above points, the decision was made to use Python for developing the lightweight multi-agent system framework and C for developing the gait identification algorithms. The interfacing mechanism in Python is then used to embed the gait identification algorithms into the multi-agent system framework.

6.4 Communications Protocols

The protocol best suited for the proposed multi-agents system is TCP (Transmission Control Protocol) [44], which has a number of key characteristics:

- *Reliable (error-free)* : Data is delivered in the right order, and there are no duplicates.
- *Connection-Oriented*
- *Two-Way Communication*
- *Slow*: Compared to UDP (User Datagram protocol) [43].

The alternative to the TCP protocol would have been to use UDP (User Datagram Protocol). In UDP protocol the data packets are sent one after another and there is no checking to see whether they have arrived at the destination and for that reason it is a very fast protocol.

In this application, the reason why TCP was preferred over UDP Protocol was that, in the agent architecture, the reliability of the message delivery is a very important issue. Therefore, it was important to choose a protocol which guarantees a safer delivery of the messages between components as opposed to a faster one.

There is scope within the system to add support for UDP connections, but there would have been more work to do in implementing the system. All the data being sent would have to be manually verified as there is no guarantee that there is no packet loss or duplication or even that the packets arrive in order. TCP handles this automatically, so for the initial development of the system it was the best protocol to choose.

6.5 The Implementation of the Lightweight Multi-Agent System Framework

6.5.1 Agent/Worker Implementation

The Agent/Worker base class is one of the core components of the system. Almost every other component of the system is a derivative of this class.

The functionality of the agent/worker is simple. When the agent starts up, it launches a thread to deal with the Communications between the agent and the router and, optionally, one to deal with user input from the command line. Once these threads are running, it subscribes to the router.

Agent-Router Communications

The majority of the work done by the base agent class is communications based. As detailed before this can be looked upon in two separate sections:

- *TCP/IP Communication Layer*
- *Message Parsing Layer*
- *Message Transmission Format*

TCP/IP Communications

Communications in a multi-agent system need to be reliable and, as discussed before, TCP connections are the most suitable for this task. In this system an agent can receive arbitrary length messages from the router, at any time. Additionally there is no guarantee due to the nature of TCP communications that we will receive only one complete message every time we get more data. For this reason it was decided to implement a buffered asynchronous communications system.

Due to the asynchronous nature of the communications model, the agent must be constantly listening for incoming messages. As such the listening must be done in its own thread, separate to the rest of the agents' functionality.

In a fully buffered communications model, both incoming and outgoing data would be buffered. However, the data being sent makes little sense to anything other than an agent of this system, all of which are required to buffer their received data. Hence, buffered output is not critical and has not been implemented to save complexity and execution time.

- *Sending Messages*: To send a message using the TCP protocol the agent creates a TCP socket. Each TCP socket has two built-in functions for sending and receiving the messages.

  The agent's send method uses the built-in socket library's *send* method. This method when called by the socket object returns the number of bytes sent. We then check that all data has been sent. If only some of the data was transmitted, a further attempt is made to deliver the remaining data.

  In order to use the socket's `send` method, our socket must be connected to a remote socket.

- *Receiving Messages/Listener Thread*:

  When the agent is started the first thing it does is starting a listener thread to receive any message that it might be sent at any time.

  The listener thread is a simple section of the agent contained within the "listen" method. Its jobs are as follows:

- wait for, and accept a connection from the router following a subscription request.
- receive data from this connection and place it directly into a receive buffer.
- check for a disconnection and act upon any premature closure of the connection by the router.

Hence, its first job is to create and bind a socket on the local machine, then to call the socket's `listen` to wait an incoming connection. The `listen` function will block any further execution until a connection is requested. At this point we can "accept" the connection, storing the connection and its address in "conn" and "addr". The "conn" object is later used to send the data.
Next, the function enters a while loop that it will only exit once it stops listening. In this loop, a constant call is made to the `getmessage` function. This function performs a buffered read of data from the socket in chunks of up to 1024 bytes at a time.

After each chunk of data is read the data is appended to the end of a buffer. The buffer is then checked to see if an entire message is present. If "yes" then the message is removed from the buffer and it is returned by the `getmessage` function.

The `getmessage` function will then continue to listen for/receive data until either an error occurs or an entire message is read into the buffer.

The message returned from `getmessage` is checked to see if it is a "disconnect" message. If "yes", no further processing is performed and the agent is disconnected and stopped. Otherwise, the message is added to a queue waiting to be processed by the main agent's thread.

- *Subscribe*: Once the listener thread has been started, the agent then subscribes to the router and listens for further messages. In the `subscribe` method, the agent first creates a unique ID which is a simple combination of its name, port numbers and IP address. It then constructs a "subscribe" message. The messages in this multi-agent system are implemented in Python using dictionaries, which are essentially associative arrays. This allows for simple parameterization of the data contained within the messages. The communication language for creating the messages in this system is a custom-made one highly inspired by FIPA ACL which was briefly described in 3.5.3. Messages are transmitted using TCP (Transmission Control Protocol).

  A typical message in this system is of the following format:

  - `performative`: Explains what the message is about.
  - `target`: The name of the agent that the router needs to forward the message to.
  - `sender`: The identity of the agent which sent the message.

  Therefore, the "subscribe" message will have the above parameters set to the following:

  - `performative`: subscribe
  - `target`: router
  - `sender`: Agent's unique ID which is sending this subscribe message (as mentioned above this is a combination of its name, port number and IP address).

– `portno`: This is the agent's port number. This is needed by the router to know which port to connect back to.

After constructing the "subscribe" message, the agent creates a socket for connecting to the router using the available environment variables which hold the router's address (IP, port number). It then sends the "subscribe" message to the router using this socket and awaits an acknowledgment message. Once the acknowledgment message has been received, the agent then closes the socket.

Message Parsing Layer

Whenever an agent receives a message, it checks the "performative" parameter and acts accordingly, retrieving any other parameters that might relate to the command in the message.

There is a number of standard "house-keeping" commands that the base agent class will parse and process by itself (without the need to forward to others to deal with). These are detailed below:

- Kill: This command instructs the agent to terminate itself. When an agent receives this message it should exit cleanly and immediately.
- Ping: This command is used to ascertain whether the agent is still alive and responding to messages. The agent, on receiving a ping message, should return a corresponding pong message to signal that it is still active.

Message Transmission Format

Whilst the Dictionary data-type is very convenient for manipulating the data in a message, it is not in itself a transmittable format. When the message has been formed it then needs to be serialised into a single string for transmission across the network.

For this there is a comma separated list of data elements contained within a set of "}" and "{" characters. Each data entry is a comma separated pair of the key and value for one element in the dictionary, again contained within a pair of "{}" characters. An example of this format is given below:

```
{{"command","kill"},{"sender","agent1"},{"target","agent2"}}
```

This format is effectively a serialised Dictionary of the form

```
"{{key,value},{key,value}}"
```

Once received by the target agent this format must be parsed (simply by un-serialising the message) and the data placed into a dictionary object before it can be manipulated by the receiving agent.

Due to the structure of a message, the function can find an entire message by counting the number of "{"'s and "}"'s found in the buffer until there are an equal number of both found and this number is greater than 0.

The nature of the TCP protocol means that whilst it guarantees delivery of the message there are no guarantees for the message all arriving in one go. Therefore, it is possible to only

receive half the message at the receiver end. This results in some agents not responding and going into an idle state.

The solution to this is to append everything we receive into one large buffer and extract single complete messages. The state diagram in Figure 6.1 shows the process of extracting the messages from the incoming message buffer.

User Input

This thread simply parses the messages that the agent receives from standard input (stdin). In the base agent class it is only used to process an exit request from the user, allowing the agent to terminate gracefully and tidily.

6.5.2 Router Implementation

As mentioned before, the router component is an agent itself, therefore, the details mentioned in the previous section regarding the implementation of the agent/worker class, also applies to the router.

However, there are a few differences which we will look at below.

- The router starts up its listener thread on a fixed port number and IP address: These two parameters have been set as environment variables on the system. Hence, this information is available to all the agents for finding the router.
- The router does not subscribe to anything
- The router functionality differs from standard agent/worker

Router Functionality

- Agent subscription: When the router receives a "subscribe" message from an agent in the listener thread, it first checks its agent look-up table for duplicates and delete the entry if a duplicate exists. Otherwise, it creates a "RouterConnection" object which connects to the agent using the port number which is specified by the agent as a part of the subscription message. Then, the "RouterConnection" object is stored in a look-up dictionary identified by the agent's unique ID (combination of agent's IP address and port number).
- Message Forwarding: This is done as a part of processing the message buffer in each agent's "RouterConnection" object. There are two main cases which could occur as a result of processing this message:
  - Target of the message is the router: The command could be one of the following house-keeping functions such as: Kill, show or disconnect. No forwarding is needed and the relevant functions are called within the router to perform the command.
  - Target of the message is another agent: The router looks up the target in its agent look-up table and uses the appropriate "RouterConnection" object for forwarding the message to the target agent.

6.6 The Implementation of Multi-Agent Gait Recognition Batch Processor (MAGRB) Agents

This system is implemented within the lightweight multi-agent system detailed in previous section. Therefore, it is not necessary to repeat the implementation issues which relate to the framework. However, there are two specific types of agents which are implemented in the MAGRB to allow it to perform a batch processing behaviour.

These are:

- Slavedriver agent: Manages a list of tasks to be performed and assigns these tasks to the slave agents. The Slavedriver agent looks up the next task on the list and sends it to the appropriate next available slave agent. It then monitors the status of these tasks.
- Slave Agents: Agents which perform the tasks given to them by Slavedriver agent. Upon the completion of each task, these slave agents return any generated output and report back to the Slavedriver agents with the status or outcome of the current assigned task.

All communications and message passing is handled by the lightweight multi-agent system via the standard router/agent communications outlined in previous sections (see Section 4.1).

The interaction between the agents in this system can be seen in the diagram in Figure 6.2. More details are given in the following section.

6.7 The Slavedriver

6.7.1 Task List Initialisation

Similar to all the other agents, the Slavedriver begins its life cycle by subscribing to the router and establishing its communications layer in the usual manner. Once this has been done, it then begins its more specialised way of processing.

The list of tasks to perform are loaded by the Slavedriver in the form of an XML file, much like the one below:

```
<?xml version="1.0"?>
<sequence>
<data>
<path dirname="/home/incoming/GaitChallengeAuto/vid/concrete_briefcase/"/>
<task filename="03793C0ALB.vid" start="0" stop="0" action="avgsym"/>
<task filename="03792C0ALB.vid" start="0" stop="0" action="avgsym"/>
<task filename="03790C0ALB.vid" start="0" stop="0" action="avgsym"/>
</data>
</sequence>
```

The Slavedriver uses built-in routines for parsing XML files. The above format is a generic one suitable for any set of tasks to be performed by MAGRB. The Slavedriver agent parses this XML document to acquire the information needed for managing the tasks and distributing them amongst slave agents.

The "path" field in the XML, specifies to the Slavedriver where it can find the files referenced in the task list. The "task" field has a number of parameters, the name of the file

to process, frame numbers within the video file to start and stop processing from or to, and finally the algorithm to use for processing.

In the above example, the "start" and "stop" fields are set to "0". This is a special case signifying that the whole video should be processed. The "action" field shows that the algorithm to process this file is "Average Symmetry" or "avgsym".

Once the XML file has been parsed, the tasks are stored in a list within the Slavedriver, ready to be sent to the next available slave agent in the system. The Slavedriver waits until prompted to send the next task indicated by the receipt of a "ready" message from slave agents.

6.7.2 Assigning Tasks to Slave Agents

On the receipt of a "ready" message from a slave agent, the Slavedriver first looks at the *type* of slave agent which has sent the message. This is under the "sender" field of the message. It then looks through its task list until it finds the first entry where the task to be performed matches the slave agent's type.

For example, slave agent of type "avgsym" sends a ready message to the Slavedriver. The Slavedriver looks through its task list until it finds a task which requires an "avgsym" algorithm.

Once an appropriate task has been found for the slave agent, the Slavedriver calls one of its two available functions to assign to the slave agent.

The `Call` Method

The `Call` method is for simple generic task invocation. A message is sent to the appropriate slave agent containing the following fields:

- `filename`: The name of the file to be processed.
- `target`: The name of the slave agent to do the processing.
- `sender`: The name of the message sender, in this case "slavedrv".

This is a generic method and the only data related information sent is the filename. It is assumed that no extra parameters are needed in this case.

The `IP` Method

The "IP" (Image Processing) method is more closely tailored to temporal or ranged tasks, such as ones which process a section of the data. There are three extra fields which can be used in the message sent using this calling method. This makes it more suitable for tasks such as image processing than the standard `call` method mentioned in previous section. The reason is that tasks such as image processing need extra information about how the file should be processed as well as just the filename.
These fields are:

- start: A marker specifying where to start processing (in the case of a video file this is a frame number).
- stop: A marker showing where to stop processing (also a frame number).

- dataout: A field to specify the filename for the output from the slave agent's completed task.

These three extra pieces of information are vital for carrying out image processing on video files. Therefore, the IP method is most often the one chosen by the Slavedriver for sending such tasks to slave agents.

After the Slavedriver has sent a new task to the slave agent, it waits to receive a message from the slave agent indicating that it has started processing the task. The Slavedriver then adds this task to an Active Task list, along with the details about the slave agent which is processing it. Eventually, the slave agent will send a "completed" message to inform the Slavedriver that it has finished its assigned task. This task is then removed from the Active Task list. The List of remaining tasks and the Active Task list are then both checked. If there are no more tasks pending or being processed, the Slavedriver will send a "kill" message to all the slave agents, instructing them to terminate since there are no more tasks to perform.

Every time a slave agent sends a "completed" message to the Slavedriver, the Slavedriver will not instantly send a new task back to it. Instead, it will wait until the slave agent sends a "ready" message. This is because sometimes the slave agents may require some time to prepare for the new task after the current one is completed.

6.8 The Slave Agent

The slave agent is responsible for processing the data in the system is made of two main classes:

- *Main Slave Class*: a subclass of the agent class. It contains the extra functionality needed for communicating with Slavedriver for receiving and processing the data correctly.
- *Engine Class*: the part which contains the computing algorithm. It is designed to be a "plug-in" class. Once the agent is created and called to perform an algorithm, the agent base class simply instantiates and calls its Engine class, passing it the data it has received. It is then the engine class that performs the major part of the work. This means it is possible to instantiate numerous slave classes each with a different engine class to perform various algorithms.

By using both classes, a clear distinction is made between the framework side of the module and the processing side. This grants us a number of key benefits:

- Agent class itself is only sub-classed once, keeping the code a lot tidier.
- The Engine contains no code related to the multi-agent framework, hence anyone wanting to add new algorithms to the system, will only need to know about the engine class. Hence, they are not required to program any framework related functions.

In essence, the existence of engine class means that the entire multi-agent framework can be treated as a "Black Box" technology to the researchers who are just concerned with developing new algorithms for this system. This makes it easier for people to simply "slot-in" their code to this framework. Outside this multi-agent system the engine part can also be run as a standalone program (outside the framework) for testing its algorithm functionality on single files.

6.8.1 The Core Slave Class

The main slave class is derived from the class worker, and is in essence a special case of worker designed to talk solely to the Slavedriver agent. Virtually all the functionalities contained within are similar to that of the worker class.

As with all the agents in the system, the slave agents must first subscribe to the router to set up its communications with the rest.

Once connected to the system, a slave will send a "ready" message to its "master" agent. The "master" agent is override-able, however its default value is set to "slavedrv" which is the name of the Slavedriver agent. By overriding the "master" agent setting in the slave class, it is possible to make it communicate to a different Slavedriver agent. This means that one can have multiple Slavedriver agents running in the system through the same central router.

After the slave agent has sent the "ready" message to the Slavedriver, it will wait until it receives a task or is told to exit by the Slavedriver. Upon receiving a task, it sends the Slavedriver a message to notify it that it has started processing the task. It then calls the `run` method of its engine class, passing it the data which it has received from the Slavedriver. At this point, the engine takes over and performs the actual processing of the data. The slave simply waits until the `run` method has returned.

When the `run` method returns, the slave sends one of two messages to the Slavedriver. If the engine has completed the task successfully, it will send a "Completed" message back to the Slavedriver. If any exceptions are caught whilst the engine is running, the slave agent will send a message to the Slavedriver indicating that the processing was unsuccessful. The relevant error message is also logged for further reference. Following the completion of the task, successful or otherwise, the slave agent sends a new "ready" message to the Slavedriver to signal that it is ready to process more data.

6.8.2 Engines

The engine as the other major part of the slave agent, contains the algorithms required for processing the tasks. The Processing can be broadly split into two main sections: *Data Handling* and *Image Processing*.

- *Data Handling*: This part is responsible for finding the data required for processing the task and extracting the correct parts of it. This forms the input to the image processing algorithms to begin processing the task. In most cases, for the MAGRB, the data handling part opens the specified video file and reads certain frames from it into Python's image objects. These objects are then passed to the image processing part. The data handling part is also responsible for marshalling and creating the output files which the image processing functions produce.

  This part of the Engine is coded in Python, since, Python has numerous built-in functions for file handling in a platform independent manner. It also has a very rich imaging library which greatly simplifies the extracting and creating of images from DV (Digital Video) files.

- *Image Processing*: This part of the engine is the most processor intensive, working constantly on image objects pixel by pixel and performing repetitive, and often complex operations on them. For this reason, this part is coded in C, which is much more suited than Python for performing small and fast operations.

The engine's image processing parts in C are linked to the Python parts using a simple interfacing library which allows the transfer of Imaging objects from Python to an equivalent structure in C. An image object is passed on to C containing the data to be processed. Another, empty Imaging object is also passed to hold the output image.

There are a number of different image processing engines available in the system. These Engines are listed below and will be described in more detail in Chapter 5.

- Symmetry
- Period Detection
- Silhouette Generator

The class inheritance diagram in Figure 6.3 shows the hierarchy of fundamental classes in both MAGRB and MAGRD systems. This shows how the main components such as agent and slave class are used to inherit others from. This makes this framework very re-usable.

In the next chapter we will discuss the performance analysis of the lightweight multi-agent system which is the framework to hold the MAGRB system. We will also show that the proposed lightweight multi-agent system is a suitable framework for a complex software system such as MAGRB.

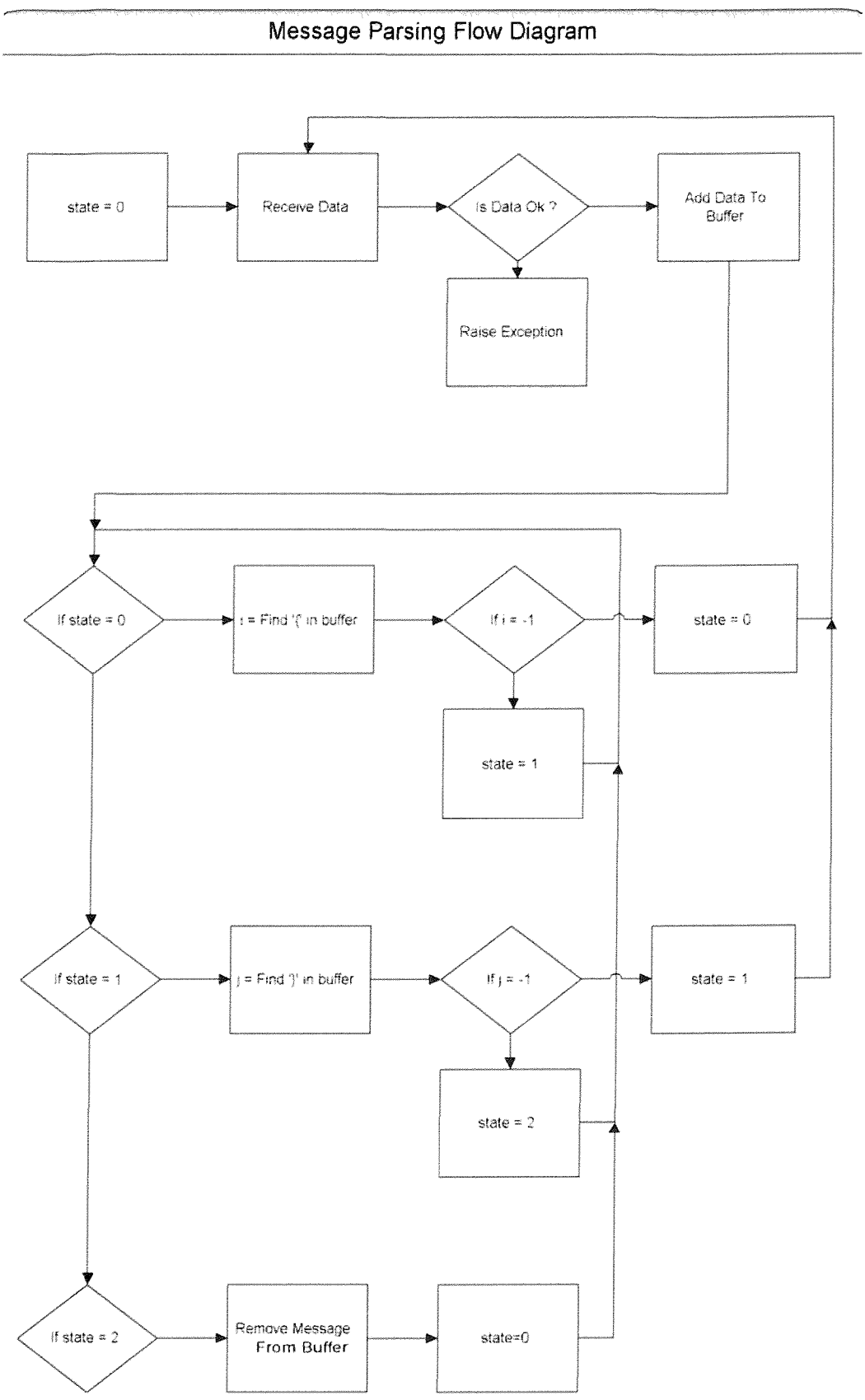

Figure 6.1: The state diagram showing the process of extracting the messages from the incoming message buffer.

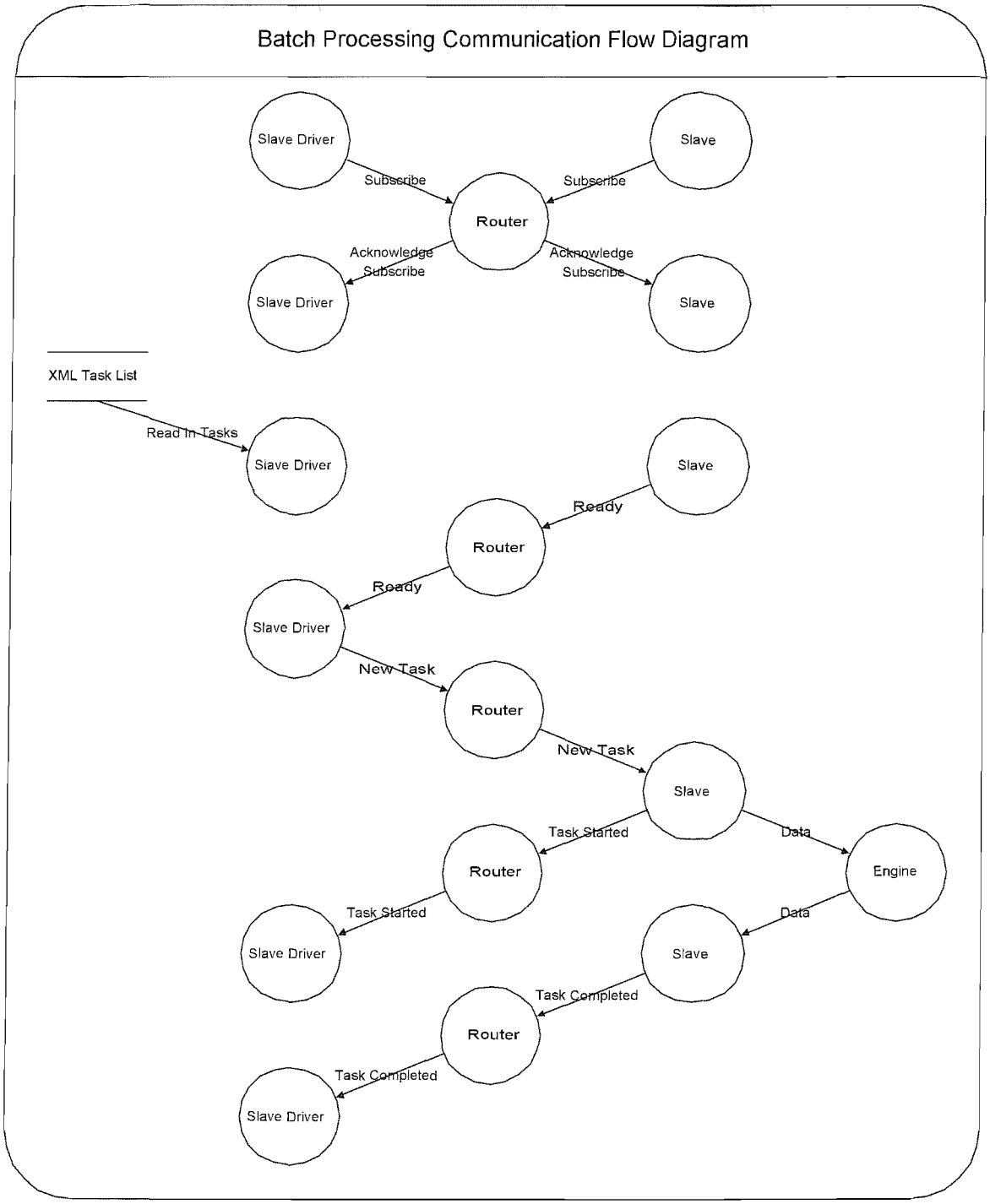

Figure 6.2: Agents' interaction diagram.

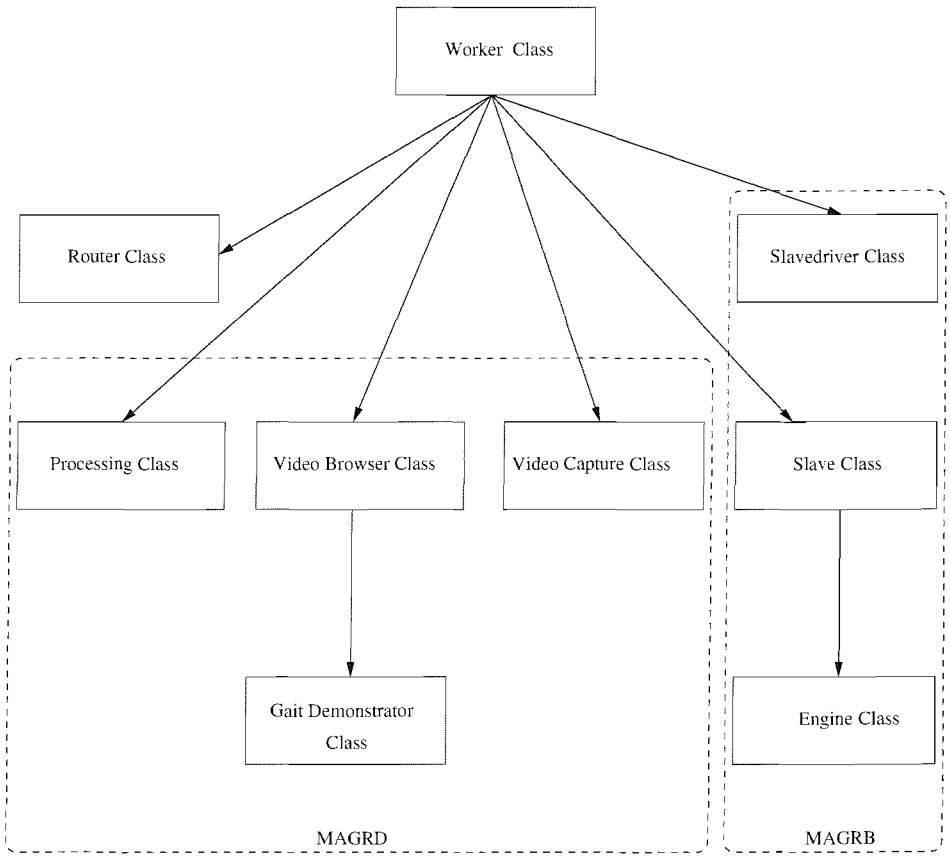

Figure 6.3: A class inheritance diagram showing the hierarchy of main classes in the proposed multi-agent system framework and how these main components are inherited from to make up the more specialised components needed for MAGRD and MAGRB systems

Chapter 7

Performance Analysis of the Lightweight Multi-Agent System Framework

This chapter looks at a series of tests performed on the lightweight multi-agent system proposed in 4 to determine its suitability for the development of the complex MAGRB system. All the following tests were performed on a single Windows machine with two P3-800 MHZ processors and 512 MB of memory.

The aim is to derive a profile for the system and to enable the user to vary the parameters involved and see the effect this could have on the overall performance. Parameters may include the reliability of the communications, speed of the network, the size of the tasks or even the file distributions and data loss. For example, if the reliability of the system is critical but the network is not very reliable, the amount of time saved by paralleling the processing is actually not beneficial to the overall performance of the system.

At the end of this chapter, some conclusions are made about how and why the proposed lightweight multi-agent system is suitable for the gait identification application mentioned in Chapter 1, such as MAGRB. which exercise the system in two main areas: scalability and robustness.

7.1 Scalability

The main aim of this series of experiments is to show how the overall performance of the system is affected by increasing the number of agents. This is done in two different ways. One is set in an ideal world where the router is performing in an optimal way and there is no delay in the way it handles the communications such as message passing. The other set of experiments show the effect of an added delay to t he router, mimicking real world conditions. We can then assess scalability by increasing the number of agents (scaling) and observing how this affects the overall performance of the system.

7.1.1 Ideal Router with No Delay

This test was performed with the router and the slaves running on a single machine. Slaves were dummies which slept for 10 seconds and did not perform any real processing. The

Slavedriver loads a task file containing 20 tasks and the experiment to calculate the total processing time for the completion of 20 tasks is run using a range of 1 slave to 20 slaves.

The aim of this experiment is to show the scalability of the system performance in an ideal world with no delay in the router. The variables are:

```
T (duration of each task): 10 secs
N (number of tasks): 20
n (number of Slaves): 1 to 20
```

Results are shown in 7.1.

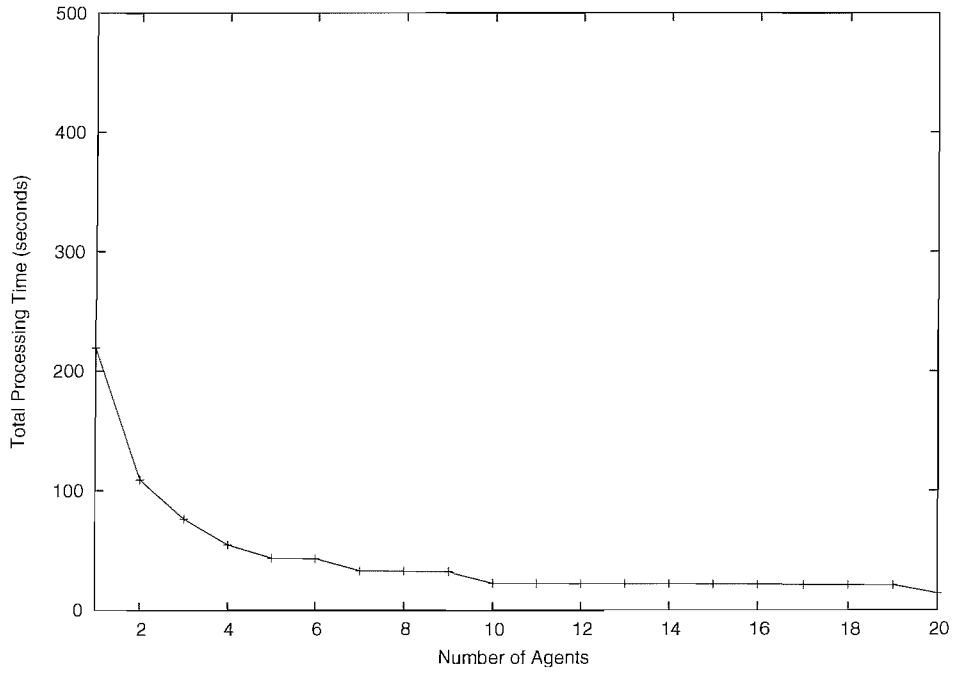

Figure 7.1: A graph showing the task processing time for an ideal router with no delay T= 10 secs, N = 20 and n = 20

Based on the fact that the total processing time is calculated by multiplying the number of tasks by the length of time it takes to process one task, then dividing by the number of agents that are processing these tasks in parallel, the results in Figure 7.1 can be formulated into:

$$T_{total} \simeq \frac{TN}{n} \tag{7.1}$$

where,

```
T is duration of each task
N is number of tasks
n is number of agents
```

7.1.2 Real-World Router with Added Delay

In real world, we can assume that there could be delays occuring in the router. These can be generated by various sources, however, due to their effect on the total processing time, they need to be factored into any formulas when trying to model this framework in a real world scenario.

The next experiment shows the effect of delay added to the router where it processes the incoming messages. The idea is to show how the performance of the router could affect the overall performance of the system. The delay is chosen to be ten percent of the total processing time for each task.

Therefore,

$$t = \frac{T}{10} \tag{7.2}$$

where,

```
T is duration of each task
t is delay in secs
```

The graph in Figure 7.2 shows the total processing time versus the number of agents used to process 20 tasks, each lasting 10 seconds. This is similar to Figure 7.1 with the difference that there is a 1 second delay in the router as opposed to 0 seconds.

The variables are:

```
T (duration of the task): 10 secs
N (number of tasks): 20
n (number of agents): 1 to 20
t (delay): 1 sec
```

The same experiment was also repeated with different values for delay to further investigate the effect of the delay on the system.

The results shown in Figure 7.3 are generated with the parameters below:

```
T (duration of the task): 10 secs
N (number of tasks): 20
n (number of agents): 1 to 20
t (delay): 0.75 secs
```

And, similarly, the results shown in Figure 7.4 are below:

```
T (duration of the task): 10 secs
N (number of tasks): 20
n (number of agents): 1 to 20
t (delay): 2 secs
```

7.1.3 Complexity Analysis

The following model is proposed to show the estimation of overall performance (task processing time) in the real world with a delay in the router.

An estimation of the total processing time in a delayed router system is calculated by adding the total processing time in a *no delay* system in equation 7.1 to the total delay inherent

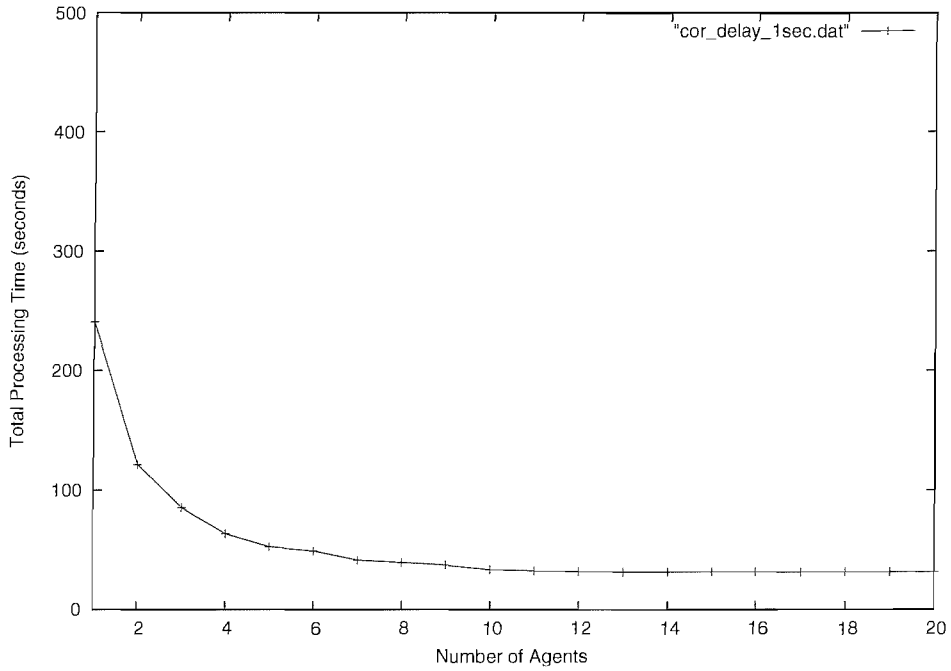

Figure 7.2: A graph showing the total task processing time for the real world router with a delay. (T= 10 secs, N = 20, n = 20 and t = 1 sec)

in the system. The delay is set for every message passing through the router, therefore, the total delay can be calculated from multiplying the delay by the number of messages per task and then by the total number of tasks. Once again this is divided by the number of agents that are processing these tasks in parallel.

An estimation of total processing time, given a number of agents in a delayed router is:

$$T_{total} \simeq \frac{TN}{n} + \frac{mtN}{n} \tag{7.3}$$

where,

```
T is duration of processing task
N is number of tasks
n is number of agents
m is number of messages
t is delay in secs
```

An agent deployed from this framework is preceded and followed by a delay. The lifetime of one agent in this delayed framework is shown in Figure 7.5.

Now, what happens if there is more than one agent in this framework which has a delayed router? The router can only dispatch to one agent at a time, which of course means that the start time for each successive agent has a delay. This, in turn, compounds the delay of the previous agent. This situation is shown in Figure 7.6

Ideally, the first agent will be given its next task by the router as soon as it has completed the first task. However, if there are more agents in the system than the optimal number, the first agent will fall into an idle state after it has completed its first task. This is because

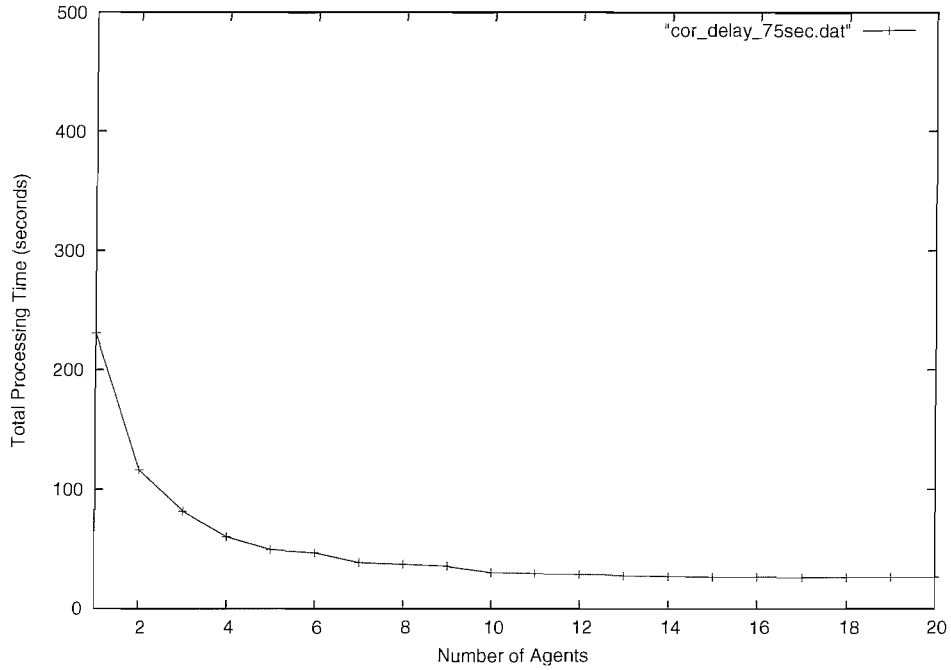

Figure 7.3: A graph showing the total task processing time for a real world router with a delay. (T = 10 secs, N = 20, n = 20 and t = 0.75 secs)

the router will be busy launching tasks to other agents which will create a small delay. This can be seen in Figure 7.7. Therefore, it seems necessary to come up with a measure for the optimal number of agents in a delayed system. It is apparent that this optimal number is dependent on length of the tasks and the delay.

As can be seen in Figure 7.5, there are two units of delay on either side of the task length. This gives us an addition of 2t to the T (where t = delay time and T = length of task). However, we need to make this factor proportional to the amount of delay in the system (this is measured in units of delay, therefore a division by delay time would make sense). This means that if the delay `t` is small compared to the task length `T`, the (2t + T) factor becomes big. Therefore, since the ideal number of agents increases, this means, we can have more agents in a framework with little delay and still gain efficiency. However, if the task length `T` is small compared to the delay `t`, this factor becomes small which means less number of agents would be more efficient. This is a very logical conclusion, since the more agents in a delayed system will add more overheads and, therefore, less efficiency is gained.

The equation for the ideal number of agents is:

$$IdealNumberofAgents = \frac{2t + T}{t} \tag{7.4}$$

This can be simplified to:

$$IdealNumberofAgents = 2 + \frac{T}{t} \tag{7.5}$$

For example, in a system with t=1 (delay) and T=3 (length of agent's task), the ideal number of agents is 2 + (3 x 1) = 5. Again, it can be seen in Figure 7.6 that the first agent

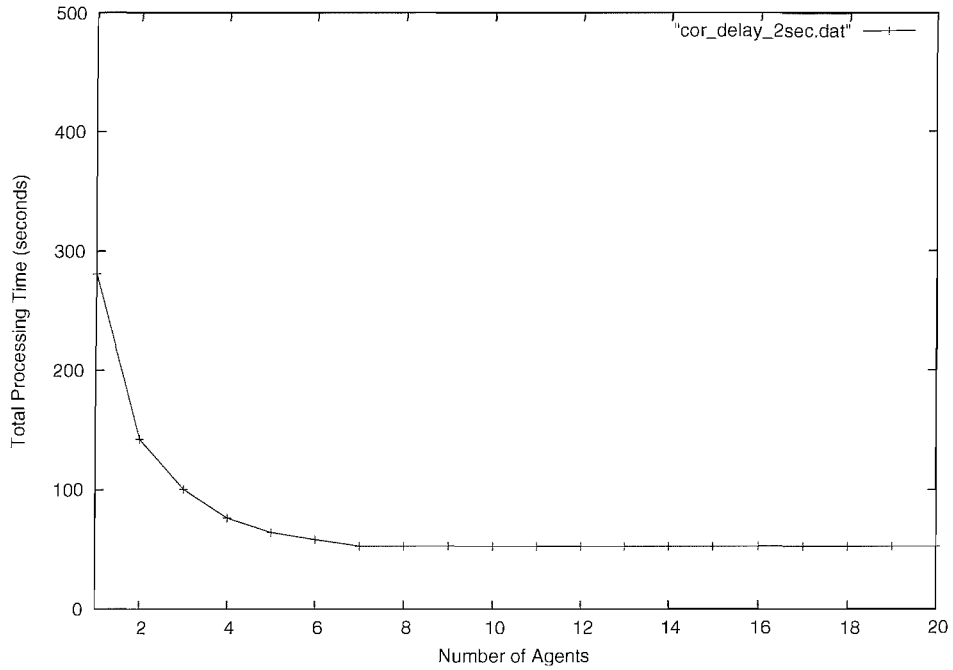

Figure 7.4: A graph showing the total task processing time for a real world router with a delay. (T= 10 secs, N = 20, n = 20 and t = 2 secs)

can start its next task after it completes its first one with only the 1 unit delay as opposed a longer lag. This is because the router would have started the 5 agents, thus it has no more agents to start up. If there were any more agents to start up, the first agent would have had some idle time whilst waiting for the router to start these up. We then need to add this observation to the model in Equation 7.3. How do we propose to do this? We need to model the effect of router inefficiency, so by finding the difference between the actual number of agents started for a set of tasks and the ideal number of agents to complete the tasks, this can mirror the router in the real world. This gives us the number of "redundant" agents in the system, ie the number of surplus agents that do not actually help to reduce the overall processing time for the system:

$$(n - (\frac{2t+T}{t})) \tag{7.6}$$

Now that we know how many "redundant" agents we have in our system, the effect of their delay in the system can be modelled by multiplying the number of these "redundant" agents by the amount of delay (t) that each one introduces into the system. This is added to the Equation 7.3 to obtain the total processing time in a real world router with delay.

$$\frac{TN}{n} + \frac{mtN}{n} + t(n - (\frac{2t+T}{t})) \tag{7.7}$$

Therefore,

$$\frac{TN}{n}+\frac{mtN}{n}+t(n-(\frac{2t+T}{t})) <= Totaltimeinrealdata <= \frac{TN}{n}+mtN+t(n-(\frac{2t+T}{t})) \tag{7.8}$$

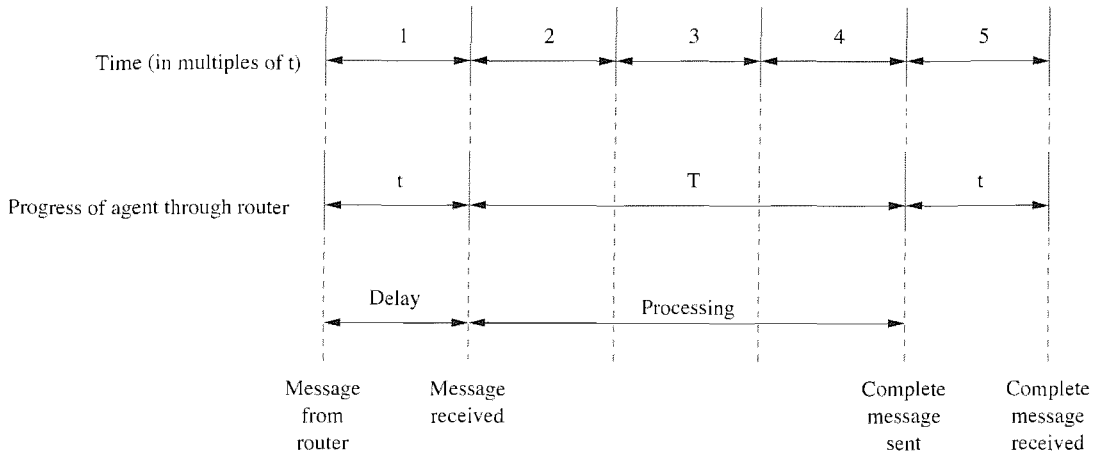

Figure 7.5: The lifetime of one agent in a delayed framework (t = 1, T = 3, n = 1)

The only difference between the lower and upper boundary is that in the lower one we divide the `mtN` component by `n` which is the number of agents. This is the best case scenario where the effect of the delay has proportionally been reduced by the number of agents.

Experiments were performed to support the credibility of the argument in 7.8 and to show how the real data fits between the upper and lower boundaries of 7.7. There are three Figures as results of these experiments which are Figure 7.8, Figure 7.9 and Figure 7.10. In each Figure, there are two graphs which show the lower and upper boundary from equation 7.8 and another which is the total processing time measured in the experiment with varying number of agents and varying the amount of delay. In the experiment to produce Figure 7.8, there were 1 to 20 agents. For each number of agents, the total processing time was measured with the parameters such as one second delay (t=1), 2 messages per task (m=2), a total of twenty tasks to complete (N=20) and task length of ten seconds (T=10). The same experiment was repeated with all same values for all parameters except the delay (t) which was 0.75 and 2 respectively. The results can be seen in Figure 7.9 and Figure 7.10. In these figures it can be seen that the real data obtained from the experiment lies between the lower and upper boundary graphs. It is also interesting to note that according to equation 7.8, the ideal number of agents for the experiment which produced Figure 7.8 would be (2 + (10/1)) which is 12. This can be seen in Figure 7.8 since after agent number 12, the middle graph (real data) reaches a lower shelf and the processing time remains the same from this point onwards. This means that there are more overheads placed on the system in handling the extra agents but for no extra gain in performance. In a similar way, the ideal number of agents for the experiment which produced Figure 7.9 and Figure 7.10 would be (2 + (10/0.75))=15.3 and (2 + (10/2))=7, respectively. A similar behaviour can be seen in these graphs, too, with the total processing time stabilizing after 15 agents in Figure 7.9 and 7 agents in Figure 7.10. The raw values that were used in producing the middle graphs of the above arguments can be seen in Table 7.1.

Therefore, t is more efficient not to increase the number of agents beyond the ideal number, otherwise, the efficiency gains will be lost.

Number of Agents	Time (Delay=1 Sec)	Time (Delay=0.75 Secs)	Time (Delay=2 Secs)
1	240.932150874	230.796125765	280.91144517
2	121.496194349	116.218449831	142.441292958
3	85.4000522413	81.6129055508	100.357952198
4	63.4122347952	60.1849019917	76.3785542576
5	52.4298814514	49.4275685876	64.3898392114
6	49.2456474217	46.9939291167	58.2388588494
7	41.3874872365	38.6863830459	52.8104767505
8	39.2880338144	37.0750492032	52.7898369765
9	37.2085161662	35.4531025591	52.7822220422
10	33.5073485597	30.4155782369	52.5588886297
11	32.4693428913	29.5914849513	52.5245420857
12	31.7482775045	28.7775222829	52.5588886297
13	31.7222627457	27.9545878038	52.3876897762
14	31.6416952434	27.1923593641	52.4595234869
15	31.7025828702	26.5612182554	52.5005259048
16	31.6013325716	26.5635015065	52.5304548483
17	31.6090371821	26.6708229677	52.5376241953
18	31.6319297564	26.4154840909	52.5360935538
19	31.6467925393	26.503167302	52.7962889138
20	31.4179958626	26.2565518802	52.4277792289

Table 7.1: Raw values for figures 7.8 and 7.9

7.2 Reliability versus Overall Performance

These experiments put the system through a series of robustness tests. Graphs will be used to show how the total processing time changes with varying reliability of the system. The experimental setup will be a system, with fixed number of tasks and fixed number of agents. In various runs, the agents are killed with a different probability and the total processing time is measured. This would show the performance of the agent processing time under varying values of unreliability.

The tests are performed in two different categories. The randomness element can be introduced in terms of *when* the agents are killed and *how many* are killed. This yields a fair set of results showing how the total time it takes to perform the jobs is affected when one agent is killed in different points through the lifetime of the entire run and also when different numbers of agents are killed in the mid-point of the total runtime. This gives us six sets of results. The test uses the `Delay` slave which is a dummy agent that sleeps for 10 seconds. There were 40 `Delay` slave tasks assigned to the Slavedriver agent and 5 agents in total are started up to complete these tasks. For the *when* to kill elements, 3 points in time are chosen. These are: at the beginning, in the middle and towards the end of the run. And for the *how many* to kill element, we repeat the experiment with: one agent to kill in the middle, two agents to kill in the middle and three agents to kill in the middle.

7.2.1 How does "when" one agent is killed affect the total processing time?

As mentioned above, this includes three experiments.

- To kill one agent near the beginning.
- To kill one agent in the middle.
- To kill one agent near end.

The graph in Figure 7.11 shows how the total processing time is affected by varying the points in time when the one agent is killed.

The three main points on the graph show the total processing time in relation to killing the agent near the beginning of the run (at 22 seconds), in the middle of the run (at 44 seconds) and near the completion (at 67 seconds). If you kill an agent at point 0, this in effect means that the system will start up with one less agent which means four agents in this case. At point 90, we are at the completion, so if the agent killed at this point of time the total processing time should not be affected, since this has the same effect as if no agents were killed during the progression of the tasks.

As it is shown in Figure 7.11, the later in the progress time that an agent dies, the quicker the tasks will be finished as there will have been more time to begin with when the dead agent was performing work. If an agent dies early in the running of the tasks, then clearly the total processing time will be higher as there will be fewer agents for a larger percentage of the time.

7.2.2 How does "how many" agents killed affects the total processing time?

This category includes three experiments:

- To Kill one agent in the middle.
- To Kill two agents in the middle.
- To kill three agents in the middle.

The graph in Figure 7.12 shows how the Total processing time is affected by varying the number of the agents that are killed.

It is fair to assume that through out the progress of the system when an agent dies, there are fewer agents left to carry out the rest of the work, so it will take longer to finish. The results in Figure 7.12 clearly support this theory. However, in a robust system, as long as at least one agent is still running the system is guaranteed to finish all tasks eventually. It will just take longer to do so.

From the above categories of experiments it can be seen that the total time will always fall somewhere between :

- The lower limit = the total time running with all agents.
- The upper limit = the total time running with the number of agents the job completes with.

The graphs show that the proposed lightweight multi-agent system in this work falls within these boundaries and completes the tasks even when some agents are killed. Hence it is quite a robust system.

7.3 Conclusion

From the above experiments, the main conclusion is that the proposed lightweight multi-agent system is a suitable solution for developing the gait identification system outlined in Chapter 1.

The reasons are the following:

- Although based on graphs shown, the total agent processing time increases in the presence of a real world router, this is because the delay is 10 percent of the total processing time. In the gait identification scenario, agents spend a large amount of time on processing the gait algorithms, therefore, the delay will be comparatively small and the overall effect of decreasing performance through the presence of delay will not be an issue. Therefore the system is scalable.

- The motivation behind this work is not to build a system to address such problems as the computer cluster crashing. Therefore, the robustness against the computer network dying is outside the scope and objectives of this work.

- The important reliability issue is that of individual algorithm agents dying. With the robust version of the multi-agent system, dead agents can be detected using a time-out facility in the new version of Python. Once a dead agent has been detected, the Slavedriver will re-submit the job which was never completed by the dead agent to another one. This way all jobs will be completed and none will be lost.

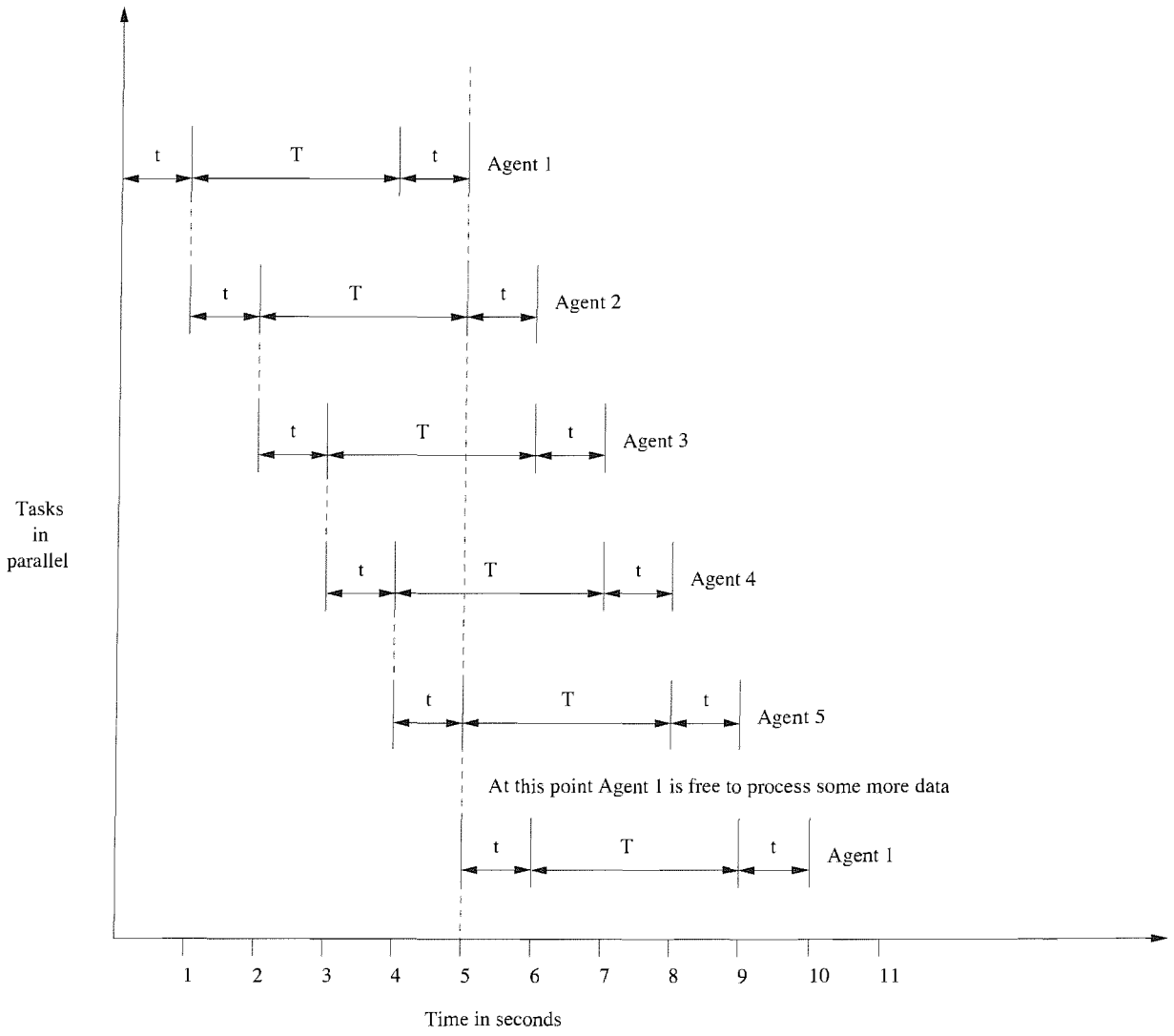

Figure 7.6: The lifetime of five agents in a delayed framework. Parameters are t = 1 (delay), T =3 (length of agent's task) and n = 5 (the number of agents))

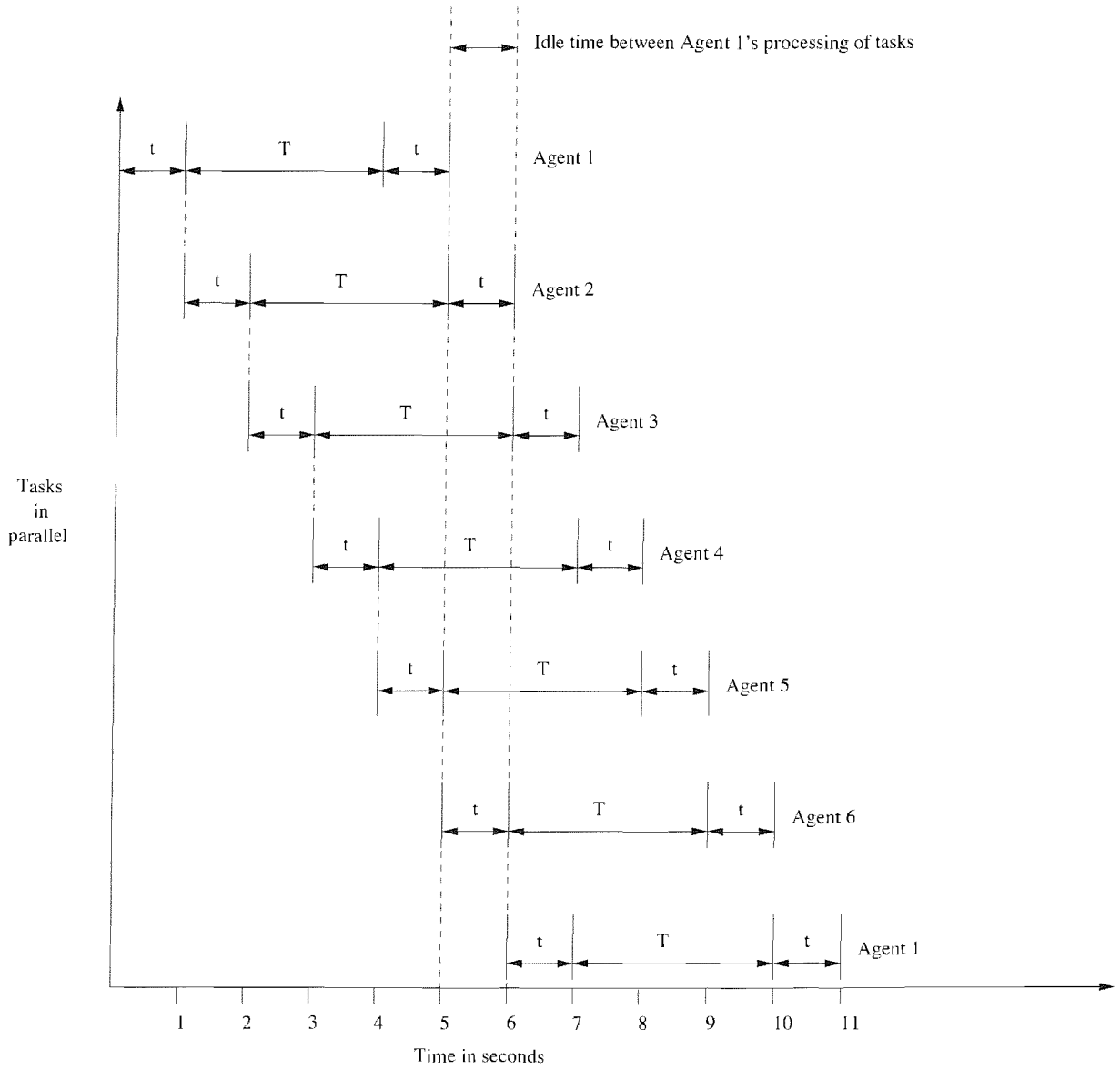

Figure 7.7: The idle time for the agent number 1 if there are more than ideal number of agents in the system. In this example this would be six agents instead of five. Parameters are The parameters are t= 1 (delay), T=3 (length of agent's task), n=6 (the number of agents)

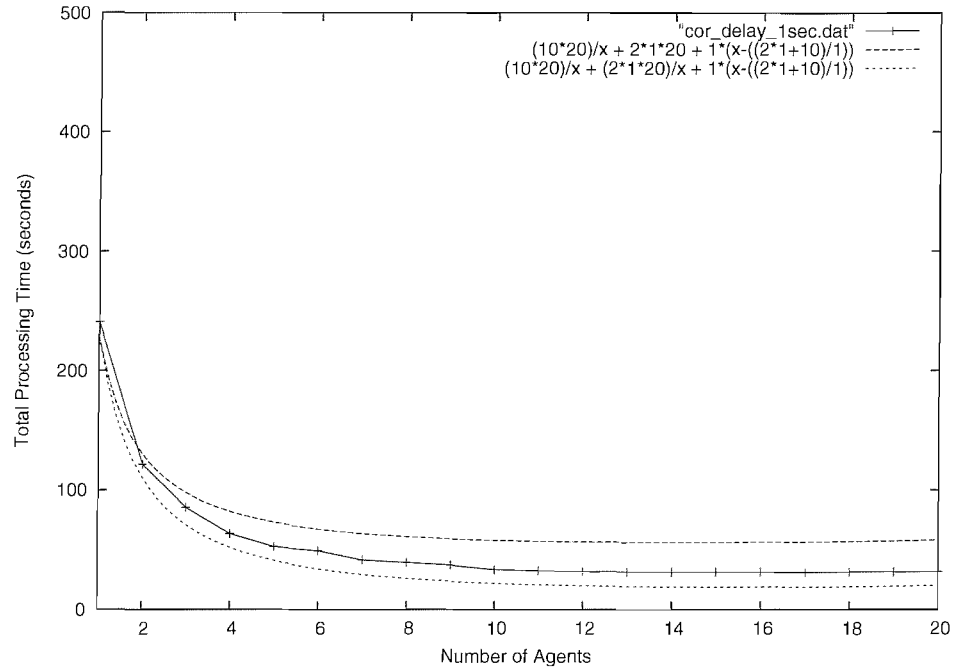

Figure 7.8: The graph showing how the experimental results fit between the upper and lower bounds of equation 7.8 of the proposed model. The parameters are t= 1 sec (delay), N = 20 (number of tasks), m = 2 (number of messages per task) and n = 20 (number of agents)

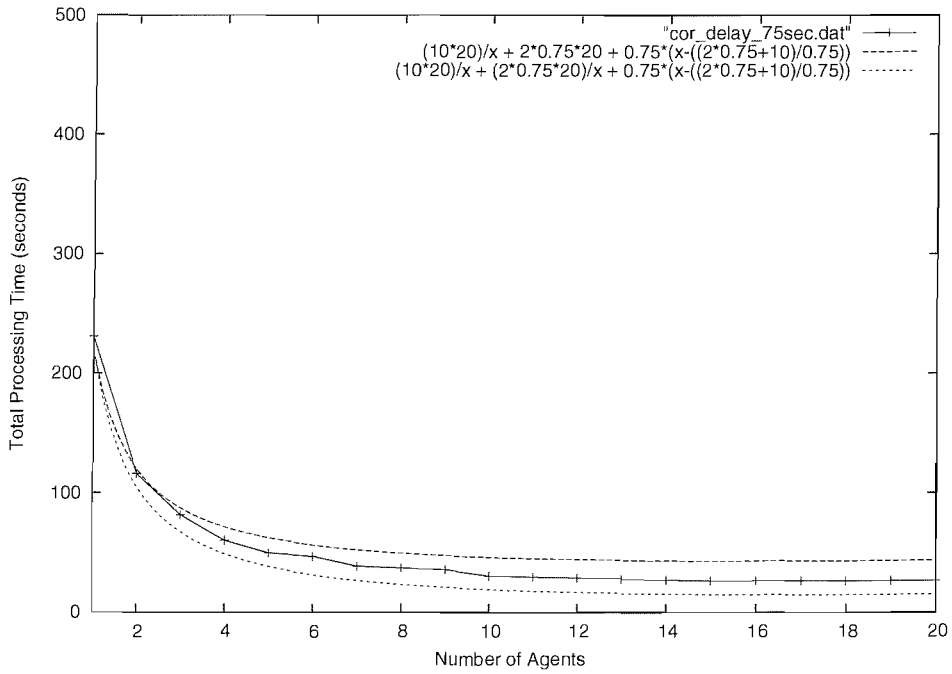

Figure 7.9: The graph showing how the experimental results fit between the upper and lower bounds of equation 7.8 of the proposed model. The parameters are t = 0.75 secs (delay), N = 20 (number of tasks), m = 2 (number of messages per task) and n = 20 (number of agents)

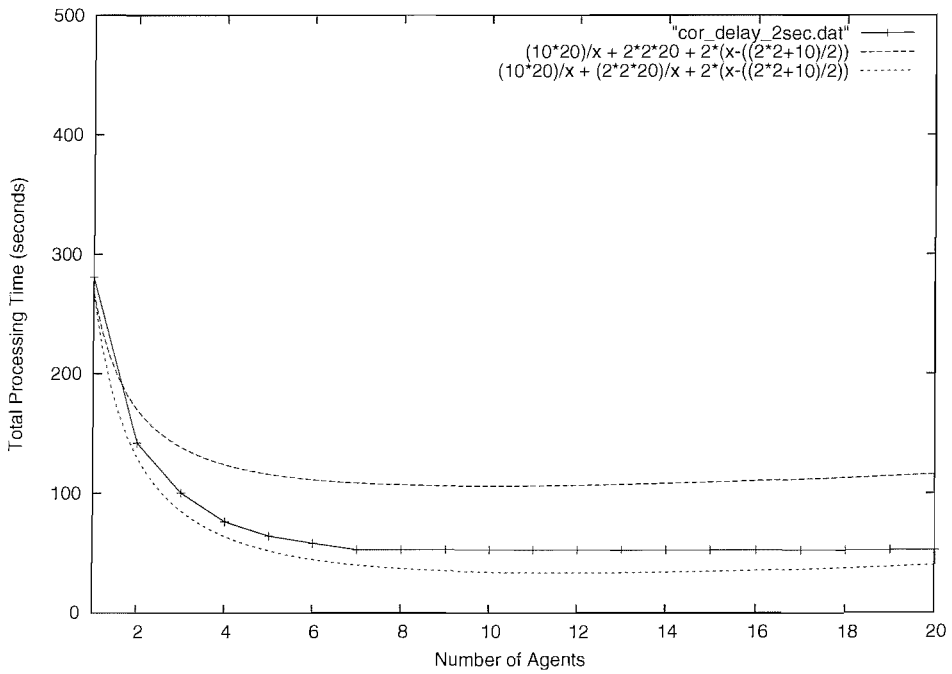

Figure 7.10: The graph showing how the experimental results fit between the upper and lower bounds of equation 7.8 of the proposed model. The parameters are t = 2 secs (delay), N = 20 (number of tasks), m = 2 (number of messages per task) and n = 20 (number of agents)

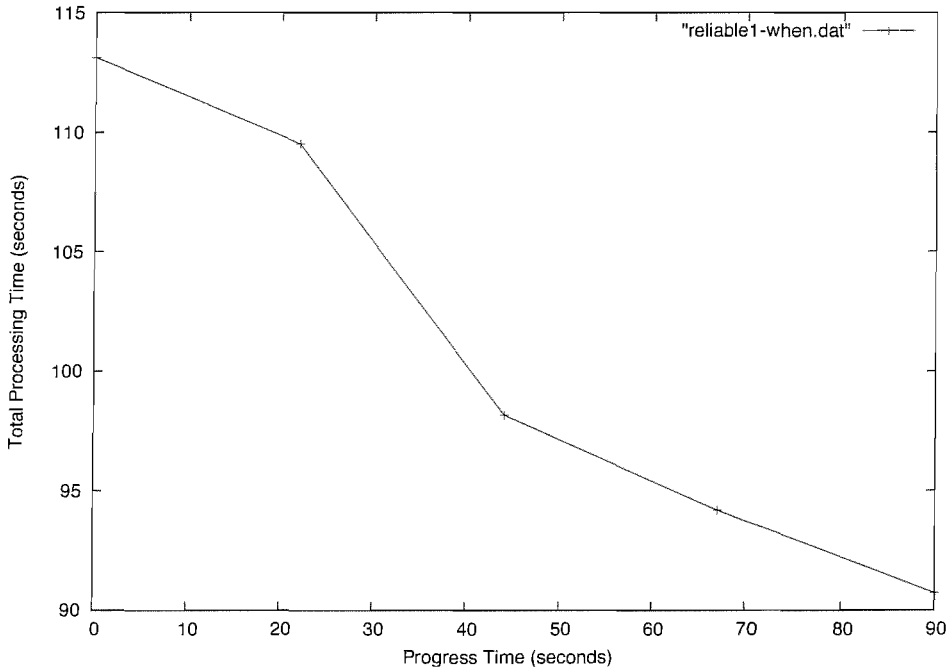

Figure 7.11: The graph showing how the total processing is affected by changing the point at which the one agent is killed. This is a reliability test run with 5 delay agents (a dummy type of agent whose task lasts 10 seconds) and there are 40 tasks in total to be completed. The agent was killed at 3 different points in the run. Near the beginning of the run (at 22 seconds), in the middle of the run (at 44 seconds) and near the completion (at 67 seconds)

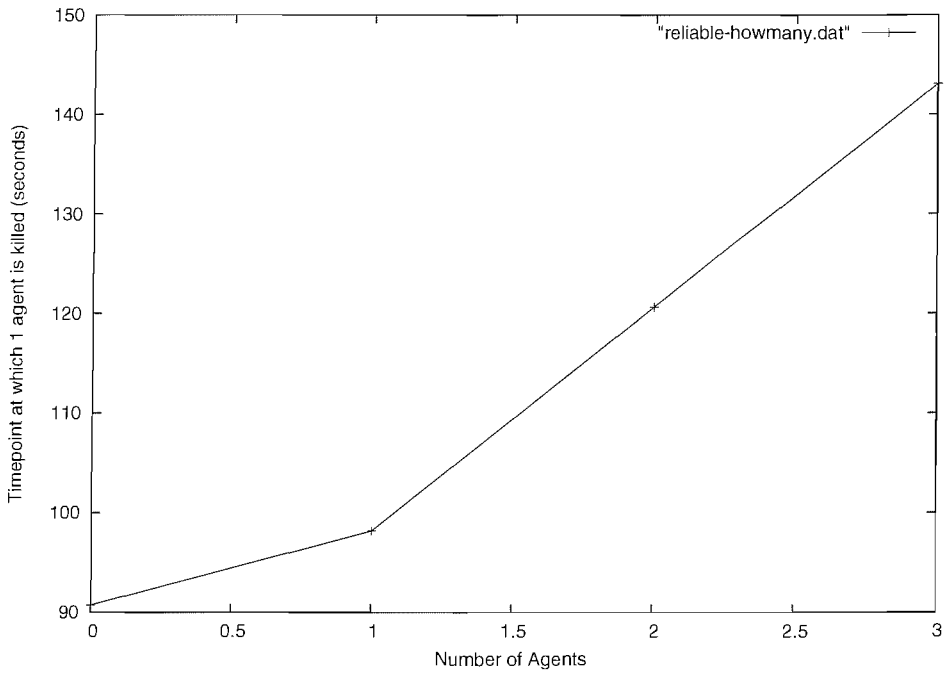

Figure 7.12: The graph showing how the total processing is affected by changing the number of agents that are killed. This is a reliability test run with 5 delay agents (a dummy type of agent whose task lasts 10 seconds) and there are 40 tasks in total to be completed. The ID numbers of the agents killed are 1, 2 and 3.

Chapter 8

Performance Analysis of Gait Identification Applications Built on the Lightweight Multi-Agent System

In this chapter, the results are shown from running the Multi-Agent Gait Recognition Batch Processor (MAGRB) system in a simple test scenario with large volumes of gait databases.

8.1 Performance Analysis of MAGRB (Multi-Agent Gait Recognition Batch Processing) System

In this section, the performance of MAGRB was evaluated in respect of running a simple test harness (dummy jobs) and on processing two large volumes of gait databases from Southampton University and the University of South Florida (real jobs). The first test intends to show the overheads of running MAGRB whilst performing dummy tasks and the second looks at its performance whilst running "real" tasks.

8.1.1 Dummy Tasks' Performance

A simple scenario was designed for analysing the performance of MAGRB's router/slave/slave driver framework.

This involved starting up a router agent, a Slavedriver agent and a slave agent on a single laptop. The slave was not performing any data processing, therefore, the CPU and memory usage shown in the following table is purely reflecting the usage of the resources by threading and communications between the key components of MAGRB. The only thing that the slave does is that it sleeps for 1 second. The laptop is a Toshiba Tecra M2, 1600 MHZ (Mega Hertz) processor, 1 Giga Byte RAM.

There are 10 tasks to be completed and there is only 1 slave running. Each task is only a sleep command for 1 second.

The graph in Figure 8.1 shows the memory usage versus the time point during the lifeline of the 10 tasks being completed in the simple run. At time point 0, the router starts up, at

time point 1, the Slavedriver agent is started up, at time point 2, the slave agent is started up (it subscribes to the router and contacts the Slavedriver agent), at time point 3, the slave agent starts its job, at time point 13, the slave completes its job and at time point 14, the slave agents are killed by the Slavedriver.

Here it is noted that when the Slavedriver starts up, it reads an XML file of tasks to be performed. The loading of XML files in Python involves reading them into a Python object. A module called "xmlobjectify" is used which transforms arbitrary XML documents into "native" Python objects. Its goal is to produce Python objects that feel more "natural" to Python programmers than DOM objects, and that are easier to work with. However, this process consumes a relatively high amount of memory. This is why in Figure 8.1 the Slavedriver's graph has got a more significant up and down bump at the start up time point (time point 2) compared to the router and the slave agent.

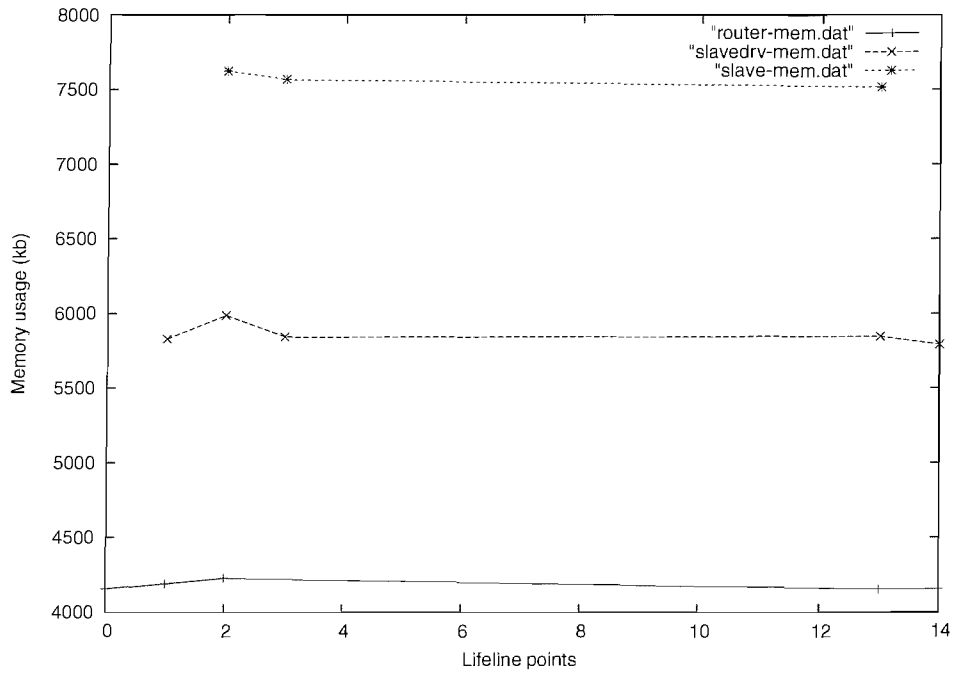

Figure 8.1: The graph showing the memory usage of the three components (router, Slavedriver and slave) in running 10 tasks each last approximately one second. The memory usage goes up at points in the lifeline of this test but generally stays low.

It is clear from the above breakdown and the graph in Figure 8.1 that at important points the memory usage is slightly higher since there are more interactions going on, but in general, with a simple use of this multi-agent system the memory usage at its highest point (which comes from the slave agent running) is around 8 MB (8000 Kb) which is very low in a system with 1 GB of memory. This is only 0.008 percent usage of memory. The CPU usage was also very low at 0.0037 percent in its highest point.

8.1.2 Real Tasks Performance

The system was used for processing two major databases. One is Southampton University Image Human ID Database and the other is the University of South Florida Gait Chal-

lenge Human ID database. There are two reasons why two different databases were chosen for assessing the performance of the MAGRB. One reason is to show that MAGRB is reconfigurable and demonstrates the flexibility of the multi-agent framework that it has been built on. Therefore, the system is not tied to a particular dataset and can be used for processing others too. The other reason is that these two databases vary in size and this would put the MAGRB system to a fair test of its performance. A brief description of both databases and the performance results of our agent framework on processing these datasets are given in the following sections.

University of Southampton Database

The datasets

This database is divided into 3 different categories:

- Large Database
- Small Database
- Temporal Database

The data was collected in the Gait lab at ISIS (Image, Speech Intelligent Systems) over the period of 15th of May 2001 until 16th of September 2002. There are 117 subjects in this database and it consists of people walking in front of the camera in an inside lab (controlled environment) on a straight line track and outside (real life environment), each using multiple camera views.

The first dataset collected was the large database and it included all of the 117 subjects, then the second one, which is small database, was collected filming 12 of the subjects from the large database introducing co-variances such as varying shoes and clothing.

The third dataset, which is the temporal, was filmed using the same subjects in the small database but in several sessions which meant that there is a time variation to be assessed over days, weeks and months.

The total size of the Southampton's DV (digital video) files' database is 440 GB. The number of DV files in each database for normal view (subject with a perpendicular degree to the camera) of laboratory data are as follows:

- Large Database: 2163 files.
- Small Database: 3178 files.
- Temporal Database: 1560 files.

This shows the volume of the data to be processed and the importance of having a scalable framework which allows the fast and optimised processing of large volumes of data on a networked cluster of PC's.

Processing

As mentioned before, there is a chain of pre-processing and gait algorithms to perform in order to produce a subject's gait signature to be used for recognition. The MAGRB was used for processing the Southampton database mentioned in the previous section.

As an example, Silhouette Generation is one of the first processes performed on raw data to produce the input for gait algorithms. In order to see the importance the amount of time

which can be saved by using MAGRB, some Figures regarding the algorithms run-times are given below:

In a typical indoor laboratory data, there are approximately 60 frames of size 720 × 576 pixels at 24 BPP (bit per pixel) in each video sequence. Therefore, The total size of the sequence is approximately 1.20 MB. It takes 220 seconds to produce binary silhouettes of size 720 x 576 at 1 BPP from a complete video sequence of 62 frames. This is on a machine with 2 P3-800MHz processors and 512MB of memory. On the same machine, it takes 20 seconds to run period detection (see 5.3) on a typical sequence of 62 frames. Once the gait period is determined, it takes 17 seconds to run the average symmetry algorithm (see 5.4) on a gait period of for example 25 frames.

The compute cluster in the gait lab is comprised of 22 machines. They are of the following specifications:

machine	Processor	speed	memory
cluster nodes 0-7:	Athlon	1.2GHz	1GB
cluster nodes 8-14:	Athlon	1.5GHz	1GB
cluster nodes 15-22:	Athlon	2.1GHz	1GB

Table 8.1: Specifications of machines in ISIS Compute Cluster

The table below shows how long (in seconds) each stage of the gait recognition takes to process 1 sequence versus 2163 sequences (large dBase, normal view).

	single sequence	large db(serial)	large db(parallel)
Silhouette Generation	180	389340	9900
Period Detection	20	43260	1500
SAS and symmetry	17	36771	1238

Table 8.2: Statistical measures of processing time in various stages of gait recognition

There are advantages gained from parallel processing in the design of this framework. For example we know that different sequences can have different lengths, therefore, when one machine finishes a single job, it immediately gets assigned another job rather than having to wait for other machines to finish their current jobs, this would result in all machines not doing exactly the same number of jobs at the end, which also means that the faster machines will process more jobs spreading the workload as efficiently as possible.

This raises the question, is there an overhead from running the router and communications between components for the optimisation in speed that we have gained?

To answer the above question, we can look at the results of running the top command on the machine which the router is running on:

```
USER       PID  CPU MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
lg2      16487  0.0  0.0  2220  936 pts/1    S    16:56   0:00 ./run_forever.sh
lg2      16490  0.0  0.3 56540 3980 pts/1    S    16:56   0:00 python startup.py 0
lg2      16495  0.0  0.3 56540 3980 pts/1    S    16:57   0:00 python startup.py 0
lg2      16496  0.0  0.3 56540 3980 pts/1    S    16:57   0:00 python startup.py 0
lg2      16497  0.0  0.3 56540 3980 pts/1    S    16:57   0:00 python startup.py 0
lg2      16498  0.0  0.3 56540 3980 pts/1    S    16:57   0:00 python startup.py 0
```

```
lg2      16499  0.0  0.3 56540 3980 pts/1    S    16:57   0:00 python startup.py 0
lg2      16500  0.0  0.3 56540 3980 pts/1    S    16:57   0:00 python startup.py 0
lg2      16502  0.0  0.3 56540 3980 pts/1    S    16:57   0:00 python startup.py
```

As seen in the above output, the CPU usage remains 0.0 and the memory usage at its highest is 0.3. Therefore, the answer is "No" since the CPU and memory usage of router (startup.py) remains low.

And below the top command's results are given for the machine that one slave agent and the Slavedriver agent is running on:

```
PID USER     PRI  NI  SIZE  RSS SHARE STAT (percent)CPU (percent)MEM
20654 lg2      9   0 12224  11M  2792 S     0.0  2.3   0:00 python
20655 lg2     16   0 12224  11M  2792 R    97.6  2.3   0:16 python
20656 lg2      9   0 12224  11M  2792 S     0.0  2.3   0:00 python
20657 lg2      9   0 12224  11M  2792 S     0.0  2.3   0:00 python
20652 lg2      9   0   948  948   800 S     0.0  0.1   0:00 sh
20653 lg2      9   0 12224  11M  2792 S     0.0  2.3   0:00 python
20648 lg2      9   0  4548 4548  2112 S     0.0  0.8   0:00 python
20649 lg2      9   0  4548 4548  2112 S     0.0  0.8   0:00 python
20650 lg2      9   0  4548 4548  2112 S     0.0  0.8   0:00 python
20651 lg2      9   0  4548 4548  2112 S     0.0  0.8   0:00 python
20647 lg2      9   0  4548 4548  2112 S     0.0  0.8   0:00 python
```

Apart from the second row which has got a high CPU usage (97.6 percent) due to it being the engine that is running the average symmetry map algorithm, the rest of Python commands running the communications are low in using the resources. This keeps this multi-agent system framework's overheads low.

Recognition Results:

Various gait recognition algorithms were run using the MAGRB system on the Southampton Database to produce gait signatures for the whole inside normal view database (6901 sequences) and as a result the graph below in Figure 8.2 shows the performance of automatic gait recognition process over time. As can be seen in this Figure, the recognition rate drops as the time between the experiment dates increases. The 'green' plot shows the average recognition rate which stays constant and the 'red' plot shows the actual recognition rate over a period of 1 to 700 days.

University of South Florida Gait Challenge Database

Data Set:

The data was collected over four days, May 20-21, 2001 and Nov 15-16, 2001 at University of South Florida, Tampa. There are 33 subjects common between the May and November collections. The data set consists of subject walking in ellipse-shaped paths in front of the cameras. Each subject walked multiple (five or greater than 5) times around the ellipse. Out of all these sequences, the last round forms the data for the subject. This is due to the fact that by then the subject would have entered the natural rhythm of their walk after having got used to the circuit.

For each person, data of up to five variations is collected. These are:

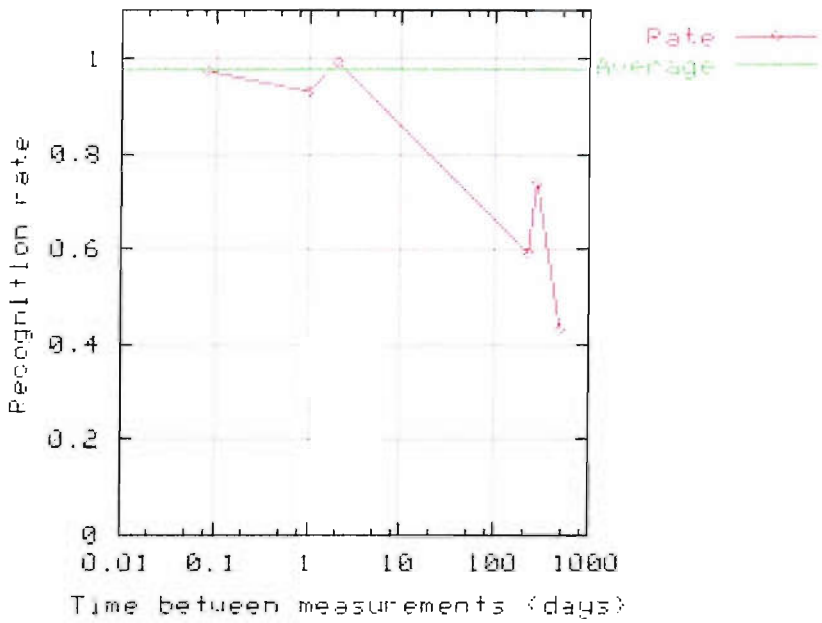

Figure 8.2: A graph showing the human gait recognition versus time covariate

- shoe types A and B for footwear variation
- two different carrying condition for subject which is with or without a briefcase
- two different surface types which is grass and concrete
- two different camera viewpoints which is either left or right
- some of the sequences were even recorded at two different time stamps

The data consists of a total of 1872 sequences of Portable Network Graphics (PNG) format images which are of size 88 × 128 at 1 Bit Per Pixel. This gives the total size of approximately 344 Giga Bytes. To find out more details about the gait challenge database, see [50].

Processing:

This dataset includes binary silhouettes of subjects, therefore, the engines that were used for processing this dataset with MAGRB were period detection and gait recognition ones such as average symmetry.

The output from the period detection was slightly different in this case from the Southampton database, because the gait signature is calculated for all of the gait periods in every sequence as opposed to just these from the most central gait cycle.

Recognition Results:

The graphs in Figure 8.3 and Figure 8.4 give the results for performance assessment of automatic gait recognition algorithms in the presence of data set Covariate explained in the previous section. The overview of the experiments used in this assessment marked with letters A to L is given in the table 8.3:

The two graphs mentioned below which show the performance of the experiments in table 8.3 are ROC (Receiver Operating Characteristic) curve and Cumulative Match Score (CMS)

Experiment	Covariate between Probe and Gallery
A	View
B	Shoe
C	view, Shoe
D	Surface
E	Surface, Shoe
F	Surface, View
G	Surface, View, Shoe
H	Briefcase
I	Shoe, Briefcase
J	View, Briefcase
K	Time (+Shoe, clothing)
L	Surface, Time

Table 8.3: List of experiments for assessing the performance of gait algorithms in the presence of Covariate

graphs. These are standard curves commonly used for assessing the performance of most biometric verification experiments.

The ROC curve plots the false positive rate on the X axis and verification rate on the Y axis. It shows the trade-off between the two rates. If the area under the ROC curve is close to 100 percent, we have a very good test. The ROC curve is shown in Figure 8.3.

Another way of assessing the performance of our experiments is by plotting the *Cumulative match score(CMS)* curve. This is shown in Figure 8.4.

A similarity score is calculated by comparing each subject's normalised get signature with other subjects' normalised signatures in the gait database. These similarity scores are ranked based on their numerical value. This means that the first similarity score will be one with the highest value.

In an ideal operation, the highest similarity score will belong to the comparison of the subject's most recent normalised signature against an existing of one of the same subject from an earlier date.

A measure called "top match score" is also calculated which indicates the percentage of times that the highest similarity score is actually a correct match for all subjects.

The other measure used is by a close look at the top five similarity scores to see if any of the top five scores are a result of comparing the subjects' most recent signature with the same subjects normalised signature acquired from the database. A measure called "Rank n' score" is then used which is the percentage of times that one of these five similarity scores is correct match for all the subjects. In this example, n is equal to five. Therefore, Cumulative Match Score also referred to as CMS (see [3]) is a plot of "rank n' scores" versus the probability of a correct subject identification.

In a similar way to the ROC curve, the bigger the area underneath the curve, the higher the quality of experiment.

By looking at both graphs, it can be seen that in both cases, experiment A (view covariate) is the best and experiment L (surface and time Covariate) is the worst.

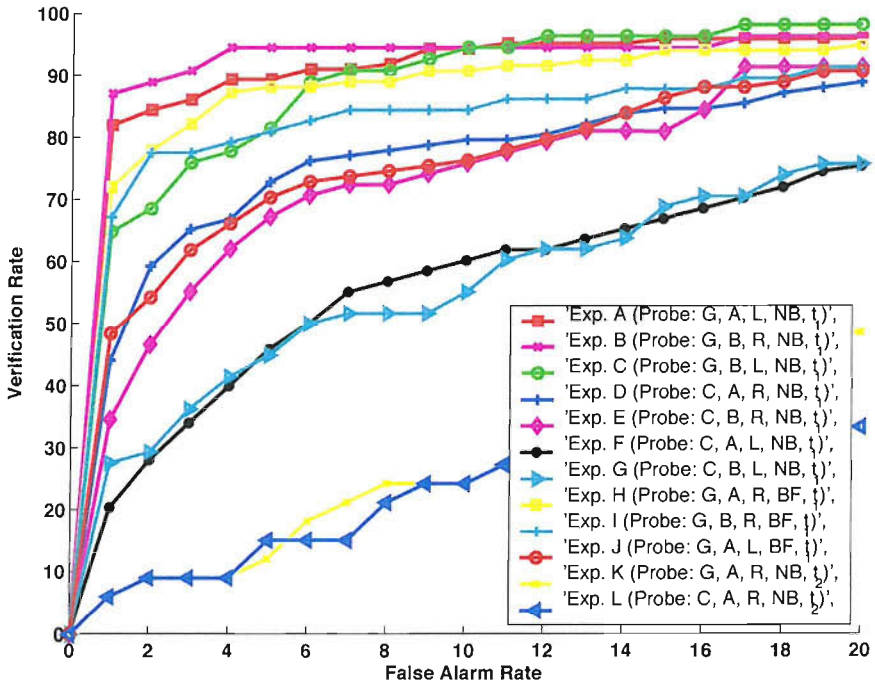

Figure 8.3: A ROC curve showing the performance of experiments A-L on covariate gait challenge data [50]

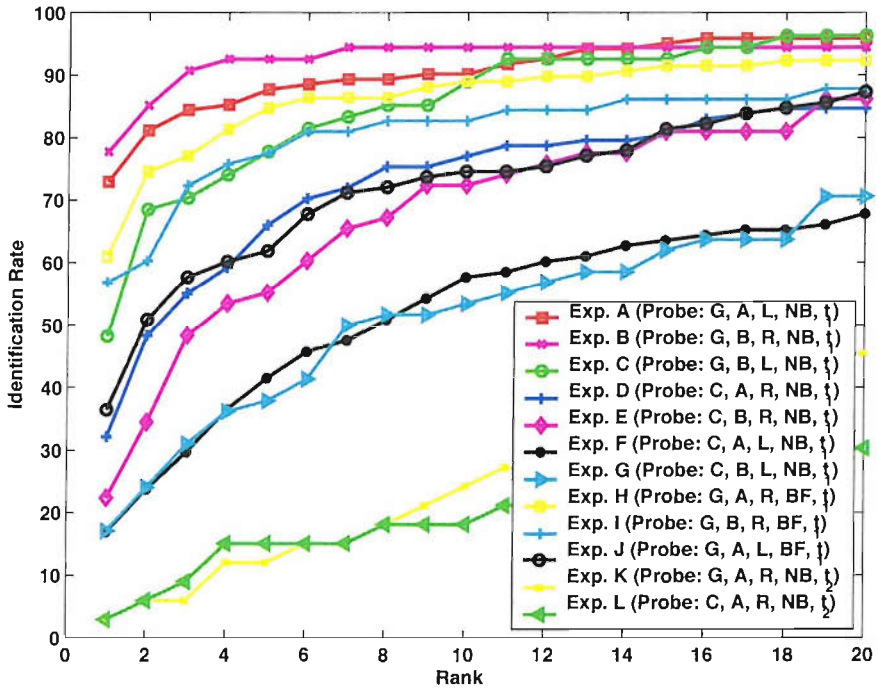

Figure 8.4: A CMS graph showing the performance of experiments A-L on covariate gait challenge data [50]

Chapter 9

Conclusions and Future Directions

9.1 Conclusions

It has been shown how the proposed multi-agent framework can be used for optimising design and build of application which are complex in terms of structure and are intensive in terms of data processing. The example scenario was an automatic gait recognition system (MAGRB), which, due to its intensiveness of data processing and a fairly complex arrangement of its components was a good example to be developed using this framework. By using this framework, the MAGRB system's data processing is paralleled, in a highly scalable manner, across a cluster of machines.

It was also shown that the communications overhead in the multi-agent framework is very low compared to data processing. This means there are very little trade-offs to be made in using this framework for building systems such as MAGRB.

Furthermore, by encapsulating the data processing algorithms purely in a concept of engines, we gain a clear separation between the algorithms and the underlying framework which makes the system generic, flexible and re-usable. Programmers updating the systems built on this framework with new algorithms, would only need to know about the how the engines operate and not necessarily the multi-agent framework. Zero or very little modification would need to be made to the framework when new algorithms are introduced. This means the framework meets the goals mentioned in 1.3

9.2 Future Directions

There is plenty of scope for improvement of this multi-agent framework. Some of these directions, if addressed, can lead to improving the overall robustness. Below is a list of some of these further improvements which can be done:

- Is there a need for a monitoring agent? This agent's main responsibility is to monitor the overall system and log all the events. There currently exists a mechanism in the framework for all the task-responsible agents such as Slavedriver to log the tasks and their status as the processing progresses. These logs can be used for tracking any problems. However, a monitoring agent can report on the status of the system in real-time whilst it is running. This gives a user a single control point for watching the system. It is worth mentioning though that in that case, there also needs to be measures in place for coping with situations if this monitoring agent dies.

- What auxiliary programs can be written for the MAGRB system to enhance its user-friendliness and functionality? One example is monitoring agent which shows the flow of processing. Other examples are a logging agent, a user utilities agent, and a visualisation agent to show the progress in a visual form including arrows for visualising the flow of communications between various components of the system.

- Should we be worried about running this framework on less reliable networks? We have assumed that network that the agent framework is running on is robust. And since the application here is not a high security one with the need for tight reliability measures, the need for programming extra parts for coping with an un-reliable network was not in the agenda. However, if this system is to be used in other applications which need a higher measure of reliability, then work can be done to cope with unreliable networks. As an example, enhanced handshaking protocols can be introduced to make the communications between the agents and the router more secure.

- Is there a need for a system with multiple router and multiple Slavedriver's? Would they need to talk to each other? The concept of having more than one router agent will be very useful in situations when the router dies. If there is another router in the system, then by using the monitoring agent, the second router can carry on managing the framework. Or if were to assume that the system is very widely distributed over not just a few machines in the same building but across the globe, the overall system can be divided into local groups of router, Slavedriver and slaves. Therefore, each local system will have their own router and Slavedriver's and in overall the routers will communicate each other. An example can be if there is an expert engine in a different domain than the one the current router is operating on, the router can contact the router of the other domain and assign a job to the other domain's agent.

- Can performance be improved by using alternative protocols? In a future system, it would be beneficial to try out other protocols such as UDP to prove the flexibility and also to see if benefits could be gained by using a better reliability and more secure protocols in the case of applications which need tighter measures.

- What advantages can be gained from designing mobile agents which can migrate their algorithms to the system where the data is? In the example of this work, the cluster of computers used for running the multi-agent framework benefited from a transparent file system, which means that the agents, no matter what machines they were running on, could access the central file stores where the data was. This might not be possible in other places and examples. Also, if the size of the data is very large, it might be more beneficial for the agents to go to where the data is and process it.

The overall conclusion is that although there is room for improvement, it has been shown that using the existing version of this lightweight multi-agent system has proved extremely beneficial for modelling and developing complex software systems such as gait identification.

Bibliography

[1] P. H. Ballard and C. M. Brown. *Computer Vision.* Prentice Hall, 1982.

[2] F. Bellifemine, A. Poggi, and G. Rimassa. JADE - a FIPA-compliant agent framework. In *Proceedings of the Fourth International Conference PAAM (Practical Application of Intelligent Agents and Multi-Agent Technology)'99*, pages 97–108, 1999.

[3] D. M. Blackburn. Face recognition 101: A brief primer. Technical report, FRVT: Face Recognition Vendor Test, 2003.

[4] G. Booch. *Object-Oriented Design with Applications.* The Benjamin/Cummings Publishing Company, Inc., 2nd edition, 1994.

[5] J. M. Bradshaw, S. Dutfield, P. Benoit, and J. D. Woolley. KAoS: Toward an industrial-strength open agent architecture. *Software Agents*, pages 375–418, 1997.

[6] J. C. Brustoloni. Autonomous agents: Characterization and requirements. Technical report, Carnegie Mellon, 1991.

[7] P. Buckle, T. Moore, Robertshaw S., A. Treadway, S. Tarkoma, and S. Poslad. *Foundations and Applications of Multi-Agent Systems: UKMAS Workshop 1996-2000. Selected Papers. Scalability in Multi-agent Systems: The FIPA-OS Perspective*, volume 2403/2002. Springer Berlin / Heidelberg, 2002.

[8] T. A. Budd. *Object-oriented programming.* Addison-Wesley, Reading, MA, 1996.

[9] M. H. Coen. Sodabot: A software agent construction system. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, page 1433, Seattle, Washington, 1994.

[10] P. R. Cohen, A. Cheyer, M. Wang, and S. C. Baeg. An open agent architecture. In *Proceedings of the AAAI (American Association for Artificial Intelligence) Spring Symposium*, pages 1–8, 1994.

[11] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[12] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 578–583, Nagoya, Japan, 1997.

[13] O. Etzioni and D. S. Weld. Intelligent agents on the internet: Fact, fiction, and forecast. *IEEE Expert*, 10(3):44–49, 1995.

[14] E. R. Fikes and N. Nilson. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[15] L. N. Foner. Entertaining agents: A sociological case study. In *Proceedings of the First International Conference on Autonomous Agents*, pages 122–129, Marina del Rey, CA, USA, 1997.

[16] Foundation for Physical Intelligent Agents. Fipa 97 part 1 version 2.0: Agent management specification. `http://www.fipa.org/specs/fipa00019/index.html`, 1997.

[17] M. Fowler, K. Beck, and J. Brant. *Refactoring: Improving the Design of Existing code.* Addison-Wesley, 2nd edition, 1993.

[18] S. Franklin. *Artificial Minds.* MA: MIT Press, 1995.

[19] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35, 1996.

[20] R. L. B French and L. Gordon. UAV 'salience' for GIS missions: Augmenting waypoint navigation with neurally processed, Gabor filter-bank outputs. In *Proceedings of GIS Research UK Fourteenth Annual Conference (GISRUK'2006).* School of Geography, University of Nottingham, 2006.

[21] D. Gilbert, M. Aparicio, B. Atkinson, S. Brady, J. Ciccarino, B. Grosof, P. OConnor, D. Osisek, S. Pritko, R. Spagna, and L. Wilson. IBM intelligent agent strategy, white paper. Technical report, IBM Corporation, 1995.

[22] I. Graham. *Object Oriented Methods.* Addison-Wesley, 2nd edition, 1993.

[23] J. R. Graham, K. S. Decker, and M. Mersic. DECAF - A flexible multi agent system architecture. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):7–27, 2003.

[24] B. Grosof, D. Chess, C. Harrison, D. Levine, C. Parris, and G. Tsudik. The role of intelligent agents in the information infrastructure. Technical report, IBM, 1995.

[25] Nwana H., Ndumu D., and Lee L. Zeus: An advanced tool-kit for engineering distributed multi-agent systems. In *Proceedings of the Practical Application of Intelligent Agents and Multi-Agent Systems*, pages 377–392, London, UK, 1998.

[26] B. Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence: Special Issue on Agents and Interactivity*, 72:329–365, 1995.

[27] J. B. Hayfron-Acquah, M. S. Nixon, and J. N. Carter. Automatic gait recognition by symmetry analysis. *Pattern Recognition Letters*, 24(13):2175–2183, 2003.

[28] M. Hu. Visual pattern recognition by moment invariants. *IRE Transacrions on Information Theory*, IT-8, 1962.

[29] A. Jain. *Biometrics Person Identification in Networked Society.* Kluwer Academic Publishers, 1999.

[30] R. Jeffries, A. Anderson, and C. Hendrickson. *Extreme Programming Installed.* Addison-Wesley, Boston, MA, USA, 2000.

[31] N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44:35–41, 2001.

[32] N. R. Jennings and M. J. Wooldridge. *Agent Technology: Foundations, Applications, and Markets.* Springer-Verlag: Heidelberg, Germany, 1998.

[33] S. Khoshafian. Insight into object-oriented databases. *Software Practice and Experience*, 32(4):274–289, 1990.

[34] H. Krasner, B. Curtis, and N. Iscoe. Communication breakdowns and boundary spanning activities on large programming projects. In *Empirical studies of programmers: Second Workshop*, pages 47–64, 1987.

[35] P. Maes. Artificial life meets entertainment: Life like autonomous agents. *Communications of the ACM*, 11:108–114, 1995.

[36] J. Martin and J. Odell. *Object-oriented analysis and design.* Englewood Cliffs, NJ, Prentice-Hall., 1992.

[37] B. Meyer. *Object Oriented Software Construction.* Prentice Hall, International Series in Computer Science., 1988.

[38] A. Newell. The knowledge level. *AI Magazine*, 2(2):1–20, 1981.

[39] H. P. Nii. Blackboard systems. *AI Magazine*, 7(3):38–53, 1986.

[40] T. E. Oliphant. Python fir scientific computing. *Computing in Science and Engineering*, 9(3):10–20, 2007.

[41] G. Oslon and S. Sheppard. Empirical studies of programmers: Second workshop. Technical report, Norwood, NJ, Ablex, 1997.

[42] I. Pitas. *Digital Image Processing Algorithms and Applications.* A Wiley-Interscience Publication, 2000.

[43] J. Postel. User datagram protocol. RFC 768, USC/Information Sciences Institute, August 1980.

[44] J. Postel. Transmission control protocol. RFC 761, USC/Information Sciences Institute, January 1980.

[45] R. J. Prokop and A. P. Reeves. A survey of moment-based techniques for unoccluded object representation and recognition. *CVGIP GMIP*, 54(5):438–460, 1992.

[46] D. Reisfeld, H. Wolfson, and Y. Yeshurun. Context-free attentional operators: The generalized symmetry transform. *International Journal of Computer Vision*, 14:119–130, 1995.

[47] J. Roberto and A. Flores-Mendez. Towards a standardization of multi-agent system framework. Technical report, University of Calgary, 1999.

[48] M. B. Rosson and S. R. Alpert. The cognitive consequences of object-oriented design. *Human-Computer Interaction*, 5:345–379, 1990.

[49] S. R. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[50] S. Sarkar, J. P. Phillips, Z. Liu, I. Robledo, P. Grother, and K. W. Bowyer. The human ID gait challenge problem: Data sets, performance, and analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(2):162–177,, February 2005.

[51] J. D. Shutler and M. S. Nixon. Zernike velocity moments for description and recognition of moving shapes. In *Proceedings of BMVC (British Machine Vision Conference)*, pages 705–714, 2001.

[52] C. D. Smith, A. Cypher, and J. Spohrer. Kidsim: Programming agents without a programming language. *Communications of the ACM*, 37:55–67, 1994.

[53] G. Veres, L. Gordon, J. N. Carter, and M. S. Nixon. What image information is important in silhouette-based gait recognition. In *Proceedings of IEEE Computer Vision and Pattern Recognition conference*, pages II: 776–782, Washington, USA, 2004.

[54] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer Society Press*, 25:38–49, 1992.

[55] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice-Hall., 1990.

[56] M. J. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley and Sons, Ltd., 2004.

[57] M. J. Wooldridge and N. R. Jennings. Agent theories, architectures, and languages: a survey. In *Proceedings of ECAI94 Workshop on Agent Theories, Architectures*, pages 1–39, Berlin, 1995. Springer-Verlag.

[58] M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering review*, 10:115–152, 1995.