A Peer-to-Peer Software Framework for Cooperative Robotic System

In Partial Fulfillment
of the Requirements for the Degree
Masters by Research

Queensland University of Technology, Brisbane
Faculty of Information Technology
June 2006
Keywords

Peer-to-Peer, JXTA, JINI, XML, RPC, WWW, RMI, UPnP, Salutation, SOAP, Instant Messaging, File Sharing, Distributed Computing, MAS, Agent
A Peer-to-Peer Software Framework for Cooperative Robotic System

by

Julie Zhu

Abstract

Recent developments in embedded systems give robots access to the Internet and make them more flexible and capable of performing more complex applications. However, these robots are still limited in terms of size, CPU power, storage resources and memory. Consequently, these robots have only been manufactured for certain specific applications and cannot be re-used for other applications. This presents us with a challenge to design a software framework - Robot Colony.

The Robot Colony enables robots to be suitable for a wide range of applications, not originally received from manufacturers, to achieve greater functionality, flexibility and utility. This research outlines the architecture and functionality of the Robot Colony to support the collaboration between devices in the P2P community and also analyse the JXTA platform, which was the framework originally proposed.

Lastly we present a customized P2P architecture that specifically addresses the interaction between software components across the network. We further discuss the following technologies applied in the framework:

- XML-based Directory Service Provider
- HTTP-based publish/describe control commands
- Remote Process Invoke

To fully complete the project, a thorough evaluation of the framework based on either the JXTA platform or the customized P2P channel has been conducted. This evaluation provides basic statistics data for the proposed framework design and implementation. Further more, we have presented a real-time Demo at the Smart Device lab of the Queensland University of Technology.
Contents

List of Figures v
List of Tables vii
Abbreviations ix
Listing xi

1 Introduction 1
  1.1 Motivation ........................................... 1
    1.1.1 Scenario 1 ....................................... 2
    1.1.2 Scenario 2 ....................................... 3
  1.2 Objectives of This Research ............................... 3
  1.3 Why the need for a cooperative robotic system such as P2P model? 5
  1.4 Requirements .......................................... 5
    1.4.1 Heterogeneity .................................... 6
    1.4.2 Network Interface ................................ 6
    1.4.3 Discovery ......................................... 6
    1.4.4 Service-driven ................................... 7
    1.4.5 Service Mobility and Lifecycle Management ..... 7
  1.5 Contribution ........................................... 8
  1.6 Significance of Study ................................ 9
  1.7 Limitations of the Study ................................ 10
  1.8 Structure of this Thesis ................................. 10

2 State of the Art of Peer-to-Peer Networking 11
  2.1 Overview ........................................... 11
  2.2 Peer-to-Peer .......................................... 11
    2.2.1 What is P2P? .................................... 12
    2.2.2 P2P History ..................................... 12
  2.3 P2P Applications ..................................... 13
2.3.1 Instant messaging .................................................. 13
2.3.2 File Sharing .......................................................... 14
2.3.3 Distributed Computing ........................................... 15
2.3.4 Other P2P Applications Examples ......................... 16
2.3.5 Summary .............................................................. 16
2.4 Related Technical Topics .......................................... 16
  2.4.1 Multi-Agent Systems ........................................... 21
2.5 Conclusion .............................................................. 23

3 Internet Robot ............................................................ 25
  3.1 Overview .............................................................. 25
  3.2 Networked Robots Controlling ................................. 25
  3.3 Robot Control Approaches ...................................... 26
    3.3.1 Direct Control .................................................. 26
    3.3.2 Supervisory Control ......................................... 28
    3.3.3 Learning Control ............................................. 28
    3.3.4 Summary ........................................................ 28
  3.4 Event-based and Behaviour-based controls .................. 29
    3.4.1 Event-based Control ......................................... 29
    3.4.2 Behaviour-based Control ................................ 29
    3.4.3 Summary of event-based and behaviour-based controls 29
  3.5 Summary .............................................................. 31

4 Robot Colony based on the JXTA platform ....................... 33
  4.1 Overview .............................................................. 33
  4.2 Introduction .......................................................... 33
  4.3 Design Goals ........................................................ 34
  4.4 Basic Platform ....................................................... 35
    4.4.1 The JXTA Structure .......................................... 35
    4.4.2 Overview of the JXTA Shell ................................ 36
    4.4.3 JXTA protocols ............................................... 36
    4.4.4 JXTA Pipes ...................................................... 38
    4.4.5 JXTA Message Passing ...................................... 38
    4.4.6 Peers and Peer Groups ...................................... 38
  4.5 Robot Colony Design and Implementation ..................... 39
    4.5.1 Functional Requirements .................................... 39
    4.5.2 Required JXTA Protocols ................................... 40
    4.5.3 Architecture ................................................... 41
## 5 Enhanced Robot Colony

### 5.1 Overview

53

### 5.2 Analysis and Conclusion

53

#### 5.2.1 Analysis

53

#### 5.2.2 Conclusion

54

### 5.3 Architecture

55

#### 5.3.1 Challenges

56

### 5.4 Requirements

57

#### 5.4.1 Requirements posed by Heterogeneity

57

#### 5.4.2 Requirements posed by Dynamism

58

#### 5.4.3 Requirements posed by P2P Network Protocol

58

#### 5.4.4 Requirements posed by Services Life Cycle

58

#### 5.4.5 Requirements posed by Services Management

59

#### 5.4.6 Design Goals

59

### 5.5 Related Work

60

#### 5.5.1 Jini

60

#### 5.5.2 Universal Plug and Play (UPnP)

61

#### 5.5.3 Salutation

62

#### 5.5.4 Conclusion

63

### 5.6 System Implementation Programming Language

63

### 5.7 Communication Layer

65

#### 5.7.1 Design of Communication Layer

65

#### 5.7.2 Implementation of Communication Layer

65

### 5.8 P2P Protocol Layer

66

#### 5.8.1 P2P Protocol Layer Design

67

##### 5.8.1.1 Communication Channel

67

##### 5.8.1.2 Service Engine

68

##### 5.8.1.3 Service Manager

69

#### 5.8.2 P2P Protocol Layer Implementation

71

##### 5.8.2.1 Communication Channel Implementation

71

##### 5.8.2.2 Service Engine Implementation

74
## 5.8.2.3 Service Manager Implementation

5.8.2.4 Configuration Facilities

### 5.9 Service

- 5.9.1 Design of Service
- 5.9.2 Name Space and Time
- 5.9.3 Service Implementation

### 5.10 Extension Services

### 5.11 Discovery and Caching

### 5.12 Conclusion

## 6 Evaluation

### 6.1 General Evaluation of the framework based on the JXTA platform

- 6.1.1 Testing Environment
- 6.1.2 Functionality Testing
- 6.1.3 Timing
- 6.1.4 Performance
- 6.1.5 Extensible Features
  - 6.1.5.1 Robot Description Caching
  - 6.1.5.2 User Interface - GUI Compatibility
  - 6.1.5.3 Advantages
- 6.1.6 Summary

### 6.2 General Evaluation of the communication Channel

- 6.2.1 Functional Checking List
- 6.2.2 Quantitative Evaluation of the Communication Channel
- 6.2.3 Comparison to the JXTA Platform

### 6.3 Conclusion

## 7 Conclusion and Future Work

### 7.1 Introduction

### 7.2 Conclusion

### 7.3 Future work

### 7.4 Summary

## A Simple P2P Network Demo

### A.0.1 User Control Panel

### A.1 Conclusion
List of Figures

4.1 JXTA Three Layers Architecture ............................................ 36
4.2 Advertising Procedure via GUI .................................................. 41
4.3 Advertising Procedure via the GUI .............................................. 43
4.4 Controlling Peer ................................................................. 46
4.5 Executing Peer ................................................................. 47
4.6 GUI - User Interface .............................................................. 49

5.1 Robot Colony Architecture .................................................... 55
5.2 UPnP Protocols ................................................................. 61
5.3 Communication Layer ........................................................ 66
5.4 An Example of Installing a New Function .................................. 70
5.5 An Illumination of Remote Control ........................................... 72
5.6 Service Manager ................................................................. 79
5.7 Service Handle ................................................................. 79
5.8 Single User Mode Illumination ............................................... 82

A.1 The Koala Robot with optional pan/tilt camera, Linux PC and wireless kit .. 101
A.2 Overview of the Demo Structure .............................................. 102
A.3 User Control Panel for the Demo .............................................. 103
## List of Tables

2.1 Comparison of Current P2P Applications ........................................... 17  
2.2 Continued Comparison of Current P2P Applications .............................. 18  
3.1 Comparison of three Control Solutions .............................................. 27  
3.2 Comparison of three control methods .............................................. 30  
4.1 Framework Functions Implementation .............................................. 51  
4.2 Framework Related Works .................................................................. 52  
5.1 Comparison of three Platforms .......................................................... 64  
5.2 Component Interface Descriptions ..................................................... 75  
6.1 Koala Specifications .......................................................................... 89  
6.2 Code Size of Different Layers of the Framework ................................... 96  
6.3 Software Components Migration Result in milliseconds ....................... 96
Abbreviations

DHCP ........ Dynamic Host Configuration Protocol
DNS ........ Domain Name System
GENA ........ Generic Event Notification Architecture
HTTP ........ HyperText Transfer Protocol
HTTPMU ....... Hypertext Transport Protocol Multicast
HTTPU ........ Hypertext Transport Protocol Unicast
IM ........ Instant Messaging
IP ........ Internet Protocol
JVM .......... Java Virtual Machine
NAT .......... Network Address Translation
OSI .......... Open Systems Interconnection
P2P .......... Peer-to-Peer
PC .......... Personal Computer
RAM .......... Random-Access Memory
RMI .......... Remote Method Invocation
ROM .......... Read-Only Memory
RPC .......... Remote Procedure Call
SLM .......... Salutation Manager
SOAP .......... Simple Object Access Process
SSDP .......... Simple Service Discovery Protocol
TCP .......... Transmission Control Protocol
TTL .......... Time-To-Live
UDP .......... User Datagram Protocol
UPnP .......... Universal Plug and Play
URL .......... Uniform Resource Location
UUID .......... Universal Unique IDentifier
WWW .......... World Wide Web
XML .......... Extensible Markup Language
## Listings

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Customized Class Loading and Creating an Instance</td>
<td>45</td>
</tr>
<tr>
<td>4.2 An Example of A Command Message</td>
<td>46</td>
</tr>
<tr>
<td>4.3 Remote Execution Code Snap</td>
<td>48</td>
</tr>
<tr>
<td>5.1 Embed an object class into a data stream</td>
<td>73</td>
</tr>
<tr>
<td>5.2 Abstract the object class from a data stream</td>
<td>74</td>
</tr>
<tr>
<td>5.3 Functional Interface Description</td>
<td>75</td>
</tr>
<tr>
<td>5.4 Description of EventHandler</td>
<td>76</td>
</tr>
<tr>
<td>5.5 Description of Event Handler</td>
<td>76</td>
</tr>
<tr>
<td>5.6 Component Bindings</td>
<td>77</td>
</tr>
<tr>
<td>5.7 Service Engine Interface</td>
<td>77</td>
</tr>
<tr>
<td>5.8 Koala Robot's XML Configuration File</td>
<td>80</td>
</tr>
<tr>
<td>5.9 Component Properties XML file</td>
<td>83</td>
</tr>
<tr>
<td>5.10 Service Interface</td>
<td>85</td>
</tr>
</tbody>
</table>
Statement of Original Authorship

The work contained in this thesis has not been previously submitted for a degree or diploma at any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Julie Zhu
June 2006
Acknowledgments

I would like to thank my supervisor Assoc. Professor Joaquin Sitte whose guidance, encouragement and friendship enabled me to undertake this ambitious research. With his support, I was able to finish this project while working as a full-time software engineer. This Masters would not have been so interesting or successful without him. I would also like to thank Erik Berglund who developed the network protocol for the Robot Colony with me and was a source of technical support.

I am grateful to my associated supervisor Assoc. Professor Paul Roe for his support and advice in every step of this project. And I also thank my previous principal supervisor Dr. Aster Wardhani for her supervision to pass through the confirmation stage of this research.

My thesis benefited greatly from discussion with my colleague Winton Campbell. His comments helped me improve the quality of the final thesis.

Many thanks to all members of the school committee to offer the fund to this project and all executive members in the school for their positive supports and for having provided all facilities that I am able to conduct this research.

Finally I would like to thank my parents and my brother. They have been, more than anyone else, the reason I have been able to get this far. I cannot express in words what their support and love meant to me.
Chapter 1

Introduction

1.1 Motivation

An astounding 98% of all processors are not in traditional desktop computer systems but in embedded devices, such as robots [EGH00]. The technology has already provided the tool for robots to access network including the Internet so that they can communicate with each other and can create a self-contained network. In this network, a group of robots cooperate with each other to achieve a common goal. Most interests in this area have evolved from a number of research groups [mit, car, ric, ias, MD].

One of the problems is that the traditional client/server network model is not suitable for this robotic system because of the system architecture nature of this model and the requirements of real-time collaboration between robots [Ora01]. The traditional client/server network model requires a central robot employed in the middle of the network and the communication between robots must pass through this server [Ora01]. In order to provide services for the group members, this central server must be loaded with all necessary code that might be required, and must be powerful enough to deal with the network traffic. Hence, the whole system’s network traffic may become blocked on the server. The network performance of this system depends on the physical network quality of this server and if there is a malfunction with this central robot, the whole robotic system cannot operate properly. Therefore, the traditional client/server network model cannot utilise the network’s three most valuable and fundamental assets: information, bandwidth, and computing resources [Gon01a].

Emerging P2P technology breaks the traditional network architecture and provides the robotic system with an alternative model. Each robot in this architecture can act as either a client or a host server, which enables robots to share resources directly with each other (the communication is decentralised). The P2P system greatly reduces network traffic and allows each robot to utilise the system’s processing power and storage capability. Hence, each robot possesses more virtual power. The P2P model is therefore more suitable for a cooperative robotic system than the traditional network model.

A number of computing researchers are evolving P2P application developments. One of these is JXTA project [jxt], conceived by Sun Microsystems, Inc. This project aims at developing a set of com-
mon P2P communication standards so that P2P applications can communicate with each other across application communities. One important feature in the JXTA platform is that it enables heterogeneous devices, including small devices, to collaborate with each other via the network. However, the interaction between robots is different from the known one of desktop computer systems in the following points:

- **Heterogeneity**

- **Service Mobility**

- **Dynamism**

- **Software Life Cycle**

This research is to investigate the JXTA platform and also to create a generic system and language independent framework - Robot Colony. This framework addresses the problems described above and provides dynamic software mobility, service discovery and remote command invocation to assist with cooperation between robots. By combining with the P2P network model, it allows robots be controlled remotely via a network. It also provides robot programmers with a facility to test, debug and modify software at the present place of work. The following two scenarios explain how the framework provides benefits to the robotic programming, testing and debugging, and how it is used to support collaboration between robots.

### 1.1.1 Scenario 1

Erik is a robot programmer. He creates a new software package and has compiled it on his computer workstation at home. Now he wants to install the software on his robot in the laboratory at the university and then test whether it works on the robot according to the design specification. The robot is equipped with a number of sensors and a camera, which can take a picture periodically.

He logs on the Internet and connects to his robot. He transmits the compiled binary code onto the robot from his computer at home. Having installed the software on the robot, Erik invokes the software using commands on the robot interface. The interface shows the measurement results from the sensors and also displays the images that the camera on the robot takes. Erik gets this result and compares them with the desired data. Then he finds a bug in his source code and needs to modify the source code. He stops the robot at the university and logs off the Internet.

After a while, Erik fixes the bugs in his source code and re-compiles the source code. He logs on the Internet and connects to his robot again. He transfers the updated executed binary code to the robot and replaces the old one with this new one. Then he enters the command on the robot interface and the interface displays the correct result this time.
1.2 Objectives of This Research

The aim of this research is the investigation of P2P applications and the applicability of the P2P network model to the robotic systems. To achieve the goal, a generic software framework - Robot Colony is developed to support the functions in these two scenarios described in Section 1.1. The development is divided into two steps. The first step is to build the framework based on the JXTA platform and the framework provides the following functions:

- Robots in the highly dynamic system must be able to cooperate with and monitor each other. This means that each robot in the system must be able to be notified by the existence of newly joined robot and the departure of a robot. Consequently, they must be able to use the available services in the system.

- The robots in the system must be able to transfer data. The data format can be a text file, a binary file, a command line message or an instant message.

- The framework running on each robot must be able to load software from other robots and run this software as its own.

- The robot must be able to call functions on other robots and get the execution results back.
The framework must provide a user-friendly GUI for operators so that they can control the robots via the network.

The second step is to develop a P2P communication channel for the framework, which supports interaction between software components on different robots. To achieve this, the following four challenges need to be highlighted:

- Robots in the system have different capabilities and are equipped with varying hardware. The software components running on these robots are software packages running on different hardware and operating systems, written in different programming language and following varying standards and protocols. To achieve interaction between these components, this high system heterogeneity must be addressed.

- Each software component is classified as a service when it is shared around the robots. Each component must be updated, modified, added and deleted for various applications without affecting the operation of other software components.

- Each software component must be able to be transferred from one robot to another robot, and must be accommodated as a native service on the robot.

- The system structure must be loose, which means each robot is free to join and leave or fail the system. Being one node of the system, the robot must be able to find the available services in this dynamic system and utilise or adopt them efficiently.

With emergence of 1) the platform independent Java programming environment, and 2) XML technology, the service description can be passed over different networks and the service can be deployed independently on the platform. By combining with the XML technology, the Java programming language and the P2P network model, the framework helps robots create an application-driven system and supports robotic cooperation (described in scenario 2 in previous section). The following design goals are finally achieved by the combination of the results from above two steps:

- Robots can communicate with each other without any limitation of the differences of network protocols, operating systems, programming languages and hardware standards.

- Robots can form an application driven system. They can either provide services or receive services in this robotic system.

- Robots can rebind the services when these services are available again after failure.

- Robots can migrate, update and delete software packages on-line without affecting the operation of the whole robotic system.

- Robots can either search for available services or enable others to find its available services.
1.3 Why the need for a cooperative robotic system such as P2P model?

There is a tendency for a range of physical devices linked together through networks (such as robots) to contain one or more microprocessors, which allows information to be collected, shared and processed in unprecedented ways [CDW00]. In such a networked system, robots are under constraints such as limited energy, bandwidth, memory and computing power limitations. The P2P system enables each node to share the resources and services directly between robots. These resources and services include processing cycles, downloading binary code and storage. The advantages of existing devices, computing power and networking connection help overcome the physical constraints of each robot, leveraging network traffic and enabling full collaboration between robots.

One important advantage of the P2P system over the traditional client/server network system is that P2P systems use the bandwidth more efficiently. The traditional network system contains one server and many robots as clients. In this architecture, the robot client can only download from the server and never upload to it. If the download path of the server is slow, the whole network performance is affected. The propagation of P2P systems is symmetric: hosts share traffic [Ora01]. If one robot’s serving side is slow, the robot still can download from other fast robots.

P2P applications can use existing bandwidth more efficiently by caching technology and controlled accessing. Distributed caches are useful tools for transmitting bulk data so that the program does not have to retransmit or resend data to another host, which reduces the sharing load. Each robot can limit the number of other robots to access its resources, which allows optimisation of bandwidth.

The P2P system is more reliable than the client/server model system, which cannot guarantee that a data packet will pass through. If the router in the traditional system is overwhelmed, it will start to drop packets at random. The P2P system behaves differently by employing the TCP protocol, with each robot in the system following the system network rules. When the network is congested, each robot’s TCP connection slows down as well in order not to lose packets.

1.4 Requirements

The aim of the software framework is to help robots dynamically form a robot group based on the application requirements. Robots in the group collaborate with each other for a common task goal. The software framework acts as an interface to individual robot resources, which allows other robots in the group to reach the resources and utilise them. The framework can also discover services available from
other robots and accommodate them into its local system. To be able to achieve the project’s objectives, the framework must meet the requirements described in the following sections.

1.4.1 Heterogeneity

The robots in the system exhibit high heterogeneity not only in their various capabilities, but also in their different requirements. Because they are from different manufacturers with various standards (including local vendor standards), the requirements at the physical level (such as hardware and operating systems) and the data level (programming languages and implementations) are different. Moreover, the communication between robots actually is the exchange of different data presentation and messages written in different languages between programs running on different hardware. Further more, each robot has a proper communication developed by its manufacturer. Hence, the framework needs to support the communication and interaction between heterogeneous robots.

1.4.2 Network Interface

The framework on the robot needs to support a network interface to allow the interaction with other robots on the network. The network interface needs to allow the robot to be:

- Controlled
- Monitored
- Programmed

The controlling interface allows other robots and operators in the system to control behaviours of the robot. It receives commands and forwards execution results to the network channel for remote method invocation. The monitoring interface allows other robots to observe and react to changes in the system. It requires the controlled robot to transfer its variables and execution results back to the controller. The programming interface allows the robot to load extra software. It also needs to be able to host user applications so that it can program, extend and control the system in ways not originally conceived.

1.4.3 Discovery

To be able to interoperate with each other, robots firstly need to detect others on the network. Robot discovery is unpredictable, as the robots do not know about other robots that they will cooperate with at the manufacturing time. Hence, the services and resources on the robots must be discovered at run time. Furthermore, services and resources will fail, join, leave or move dynamically during operation. The robot cannot bind with these resources and services at the time of compilation. Hence, the application must use currently available software components and system resources.
1.4 Requirements

1.4.4 Service-driven

The robot group is heterogeneous, which comprises of an uncertain number of robots with different capabilities. These robots provide varying services and the number of robots is determined by the application requirements. The communication between the robots is direct and the infrastructure is loose, so each robot can join in and leave (or fail) from the group freely. Consequently, each robot must adapt itself to this highly dynamic group. This dynamism implies three requirements from the framework:

- The framework must be able to recognise the available services and be able to configure its settings in order to interact with the services.
- The framework must be able to detect failed services and rebind with them whenever they are available.
- The framework must be able to adapt to new services and deploy them.

1.4.5 Service Mobility and Lifecycle Management

The robot has limited resources due to physical constraints. The purpose of robotic collaboration is to share the resources on each individual peer, to divide the task into small parts that are undertaken by robots participating together so that the whole system can complete the assigned work. Each robot in the system provides its available services for other members so that they can migrate/utilise these services as their own. These services can be either hardware (like memory) or software. However, each software package requires different arguments and data types. The robot must be able to change the configuration and parameters of its existing services and apply its services into new applications. The robot must be able to import new services and remove services that are not needed any more. Service mobility includes:

- Moving the service from one robot to another one,
- Changing the parameters and configuration of a service to suit for a new application, and
- Making the accommodated service interoperate with other robots.

The service lifecycle management provides support for installing, updating and removing a service from a robot. The requirements are:

- **Dynamism** As a node in a loose system, the robot needs to be able to find other nodes and has a mechanism to let other robots find it. Therefore, the robot needs to be able to discover and adapt to the loose system dynamically.

- **Services Life Cycle** The robots are not designed or manufactured for any particular applications and are reused for different applications, meeting the applications requirements. However, it is impossible to predict and to list all situations that a robot might face in its future work at the manufacturing stage [Tza99]. The software installed on robots tends to deal with the designed
environment. However, if there is a change in the environment, it becomes hard to control the actions of the robot.

Most robots are equipped with sensors and the present sensor integrations and these devices controlling program does not support future changes or updates in similar tasks [FSJ99]. Consequently, most existing robot systems tend to be rather static, with a very low ability to adapt to a changing environment. The existing software does not deal with new hardware components on the robot and cannot interact with any updated software that is installed.

The service on each robot can be divided into two streams. One stream includes services that come with the robot software from the manufacturer. Another stream includes services that are installed or updated later, according to the application’s requirements. Because of the physical limitations of each robot, it is necessary to remove software that have not been used so that the robot can have enough space for installing new software. The application also needs the robot to update the service to fit into the new model.

1.5 Contribution

This thesis presents the design and implementation of a P2P software framework, which allows robots to form a heterogeneous, dynamic and distributed group. The framework, referred to as **Robot Colony**, enables robots to join and leave the group freely. It allows the robot to discover available services possessed by other robots in the group. It also provides an interface to allow a robot to expose the services that it can provide to the group. It enables a robot to accommodate services from other robots in the group, according to varying applications requirements. Hence, the framework allows the robot group to efficiently operate within itself.

The thesis makes the following contributions by addressing the requirements outlined in Section 1.4:

- **Heterogeneous** The robot group is heterogeneous and consists of a wide range of robots, which are accessible and interoperate with each other. The technologies built on each robot cannot be the same, consequently their capabilities are also various [Sai02].

- **Dynamic** The thesis presents a software framework to allow robots to form a group dynamically based on the requirements from the application. The relationship between robots in the group is loose. In order to interoperate with each other, the framework provides a mechanism to allow the robots to find a robot that newly joins in the group and to find the services that it can provide. The framework also enables the newly joined robot to find available services in the group. When the robot leaves the group, the framework will notify other robots of the departure of the robot. It also provides the interface to allow the robots to interoperate with each other efficiently.

- **Network Protocol** The P2P network protocol has existed for a long time. One of its major characteristics is that every node in such a network can communicate with other nodes directly, without
a central server. Such a network protocol breaks the traditional client/server network architecture and allows every node to be either a server or a client, or both. It overcomes the uncertainty and traffic of a network under the traditional network protocols. There are many research projects in this area, but not many are applied to robot communications. According to Katia P. Sycara, agents on the internet mostly perform information retrieval and filtering. The next generation of agent technology will perform information gathering in context and sophisticated reasoning in support of user problem-solving fasts. These capability requires that agents be able to to interoperate and coordinate with each other in peer-to-peer interactions. This framework provides the robots with P2P communication ability. As such, it represents a very new area in this field.

- **Service Life Cycle** Every software embedded on the robot has a life cycle. To meet application requirements, the robot will update and rebuild software. The framework describes how the robot updates its service.

- **Services Management** When a robot joins in the group, it registers the services that it can provide so that other robots are notified by its existence and its services. When it leaves the group, its leaving does not affect the performance of the robot group. The software framework describes the way that each robot manages the services from a new robot when it joins and leaves or fails the group. It also provides descriptions of the services that a robot can present.

The thesis presents a framework to control robots remotely in a dynamic group based on the application requirements. During the execution of a task, the robot can search the service registry for a service that is required by the application and can dynamically download the service from other robots. Then it can accommodate the service into its own system. This allows the application to automate performance of the robot group.

The software framework is designed for all robots in the group and its range can be extended to other small devices. Finally, the communication model in the framework is P2P, which provides the robot programmers with direct communication to robots. This approach increases accuracy and efficiency.

### 1.6 Significance of Study

The development of robotic systems that operate in complex and highly dynamic environments has received tremendous interests from robotic researchers in recent years. In the last decade, the notion of having a team of robots cooperating to achieve a goal has gained popularity, since there are several advantages in having a group of robots cooperate to complete a required task in various applications. The type of architecture is moving from a centralized model toward a decentralized model. As the technology develops, it permits networking of different devices to form an individual dynamic network with powerful functionality.

P2P technology imposes a different view on networking - the role that network participants (peers) play when they interact and communicate, how services are provided and accessed, etc. Its flexible archi-
tecture allows us to apply the technology at different levels with varying purposes in robotic systems. For today’s problems and concerns in robotic systems, the possible solutions suggested by this technology render the P2P technology as a strong candidate for decentralized, dynamic robotic systems.

Therefore, this study will contribute to the area of P2P network in terms of the collaboration amongst heterogeneous robots, especially in the area of interoperability.

1.7 Limitations of the Study

The research focuses on those areas in P2P computing that will bring direct benefits regarding network services and collaboration. The P2P computing is a broad topic divided into four categories: instant messaging, file sharing, distributing computing and core technologies [Fox01]. The P2P technology is being applied in various areas. While this proposal is concerned with the collaboration amongst heterogeneous robots, it will focus on the design of the software framework, which enables heterogeneous robots to form a dynamic system. The framework proposed is intended for a small-size network of robots behind a firewall. The security issue is not covered within the scope of this project. Some other P2P issues, such as software certification and copyrights, are also not included in this thesis’s considerations.

1.8 Structure of this Thesis

The rest of the thesis is structured as follows.

Chapter 2 presents state of the art in P2P networking and three P2P applications. It also discusses about multi-agent systems.

Chapter 3 analyses and compares three control modes of the Internet robot. It then describes the problems faced by the Internet Robot.

Chapter 4 describes the design and implementation of frameworks based on the JXTA platform.

Chapter 5 analyses the framework on the JXTA platform and presents a solution. It also outlines the challenges for the new solution and the requirements. Then it describes the design and implementation of the framework, which uses customized P2P network interface.

Chapter 6 presents an evaluation of the framework based on the JXTA platform and the customised P2P network interface.

Chapter 7 summarises the thesis and shows the limitation of the framework. It also outlines future work for the framework.
Chapter 2

State of the Art of Peer-to-Peer Networking

This project aims to present a P2P framework that supports cooperation between robots. As P2P is the key technology in the project, this chapter will review the work done to date in the P2P area. This framework is also designed as being generic and supports service mobility, therefore it is necessary to discuss about multi-agent systems and to review the works that have been done in this area.

2.1 Overview

The rest of the chapter is organized as follows:

- Section 2.2 introduces P2P technology and briefly describes its history.
- Section 2.3 outlines three streams of P2P applications and describes the technology of these streams. The summary includes a table to compare some available P2P applications.
- Section 2.4 presents conclusions in relation to this project, including multi-agent systems.

2.2 Peer-to-Peer

Peer-to-Peer is not a new term. The early Internet was designed as a P2P mode [Ora01]. Unlike the client/server architecture, P2P networks allow peers to act as both clients and servers. Hence, in such a network, computer resources and services can be shared by direct exchange between systems [pee]. These resources and services include processing cycles, downloading binary code, and disk storage.

A P2P network takes advantage of the computing power and networking connectivity of existing devices to overcome the physical constraints of each device, leverage the network traffic and enable collaboration between devices. These features are very useful for small devices, particularly robots, which are able to access the network.
2.2.1 What is P2P?

In a traditional P2P network, computers are in a workgroup with geographic limitations and running the same networking protocols. Webopedia [webb] defines P2P as “a type of network in which each workstation has equivalent capabilities and responsibilities. This differs from client/server architectures, in which some computers are dedicated to serving the others.”

The P2P architecture of today, as proposed by Dave Winer [Win00], has the following seven key characteristics:

- User interfaces load outside of a Web browser. Any node in the P2P system can communicate with others in different networks across firewalls;
- User computers can act as both clients and servers. Each machine in the system can host services and also share resources from others;
- The overall system is easy to use and well integrated. The connection is significant, there is zero administration and the whole system is self-organised;
- The system includes tools to support users who want to create content or add functionality;
- The system provides connections to other users. The connection between users is direct and benefits for efficient networking;
- The system supports “cross-network” protocols like SOAP or XML-RPC. In order to exchange messages between nodes in different communication protocols, it requires a bridge crossing different communities.

2.2.2 P2P History

The early Internet was originally designed as a P2P system. It was much more open, symmetric and free than today’s network. The goal of the original Internet was to share computing resources, allow any two machines to co-operate with each other and to provide a group of researchers with the ability to exchange information without any protection. Usenet [Ora01], an example of an early P2P application, has been around since 1979, when firewalls were still unknown.

By 1994, more people were interested in the Internet. The internet usage exploded and there were not enough IP addresses for everybody. The network model switched to Client/Server, mostly due to the following reasons [Ora01]:

- The modem connection protocol became more common,
- Firewalls and Network Address Translation (NAT) became the common usage patterns,
- Machines did not need to have a permanent or well-known network address,
• The download model replaces the two-way model.

In the Client/Server model, clients connect to a server using a specific communication protocol to access a specific resource. The majority of the processing involved in delivering a service occurs on the server [Wil01], leaving the client relatively unburdened. Examples include FTP, telnet and World Wide Web (WWW). As the number of clients increases, the load and bandwidth demands on the server also increases, eventually preventing the server from handling additional clients. Because a client in such a model plays a passive role, they are unable to provide services to other clients.

As the Internet continues growing quickly and PC hardware develops at a similar pace, the Client/Server model becomes increasingly unsuitable. There is an untapped potential for individual machines to provide services and to exchange information directly with each other.

P2P is the key to unlocking this potential. It is driving a major paradigm shift in the era of genuinely distributed computing [Ora01]. Because every node in a P2P system can play the role of a client and a server, they can work together and share different resources. In general, P2P technologies allow people to create self-organizing end-to-end automatic communities. As Dave Winer [Win00] said, The P in P2P is People. P2P networks are part of the next wave of the Web [Fox01] and include several important features.

2.3 P2P Applications

P2P applications basically fall into three categories:

• Instant messaging,

• File sharing,

• Distributed computing.

2.3.1 Instant messaging

Instant Messaging (IM) is computer-based one-to-one communication. IM allows users to type messages into a window and to exchange the message directly with each other. Typical examples include ICQ [icq], MSN and Jabber [jab].

Like ICQ, an IM application requires a central site for initialising, getting data and connecting peers. The central server employed is also used for monitoring which user is currently online and notifying interested users. When communication between users has been established, the messages will flow directly from one user’s machine to another without involving the server.

The communicative functions of IM are [NWB00]:

• IM supports quick questions and clarifications about ongoing work tasks. IM allows rapid exchanges between users, similar to face-to-face conversation.
• Coordination and scheduling. Users can quickly learn whether the destination person is on-line when they log on to IM.

• Users are able to carry out efficient exchanges because IM enables them to eliminate certain formalities of address associated with phone and Email.

• IM contributes to greater efficiency for tasks requiring rapid responsiveness.

• IM is used to coordinate social meetings that take place face-to-face.

• IM keeps people in touch with friends and family.

As well as being a chatting tool, IM can be extended and used as a tool for Remote Procedure Calls (RPC) [tec] [sal], in which a command can be embedded inside a message and sent through the network to control the behaviour of a receiver. The receiver receives the message and extracts the command. After executing the command, the receiver sends results back in a message format.

Interoperation extends the usage range of IM. However, it imposes constraints on small devices because of communication difficulties involving different protocols and operating systems. Solutions have mainly focuses on developing a platform using HTTP, XML, SOAP and RMI, like Microsoft UPnP [tec] [sal] and Java RMI [Bac] [rmi]. However, the platform is typically too heavy for small device. Jabber [jab] provides the best solution for small devices (combining instant messaging with XML) and consequently has transparent compatibility with various IM networks.

2.3.2 File Sharing

P2P systems have two characteristics: firstly, the low cost and high availability of large numbers of computing and storage resources; and secondly, increased network connectivity.

File Sharing is a P2P application that is used to share files (most popularly MP3 files) between each node via the network. Napster [Kah, nap] is an early file sharing P2P application with centralised components. Gnutella [gnu, Rip01] advances beyond Napster without employing a central server.

• **Napster** is a P2P system that provides users with the ability to swap MP3 files. In fact, Napster mixes both centralisation and decentralisation [nap], employing a central server to store a list of MP3 files stored on each user’s machine. When a node logs on to the network, it uploads a list of all shared MP3 files to Napster server. These files are stored in a special directory at the node’s local machine. Another functionality of this server is to be responsible for allowing users to search the list of available files to find a special music file and its host. Once the user finds the host, the file transferring between them is direct, without the server. Napster also provides the facility to send messages between users.

Strictly speaking, Napster is not a pure P2P system and employs a hybrid P2P system containing some centralised components [Ora01]. Napster’s most significant contribution to P2P is that the...
central server employs a mechanism to store the pointers of individual hosts, instead of saving the whole music file.

- **Gnutella** is a more advanced P2P system than Napster. Its network is server-independent and features the ability to share any type of file [Wil01]. In order to join the Gnutella network, peers need to connect to an arbitrary host and issue a PING message. The message is broadcast to Gnutella hosts and they respond with a PONG message.

Each peer on the network is not only responsible for serving files, but also for routing messages to other peers. In order to establish the Gnutella network, there are a number of peers with static IP addresses that have to be connected so that they can provide further discovery of other peers. Consequently, the protocol of file transfer in Gnutella is HTTP. However, when traffic increases over the users’ dial-up connections, all the peers may stop passing information - Gnutella will crash and the whole network is fragmented.

The most important P2P contribution from Gnutella is a message broadcasting mechanism in which each message is assigned by a unique identifier (UUID is a 128 bit unique identifier) and runs around the network. Each time a message is delivered or originated, the UUID of the message is memorised by the host that it passes through. However, if the same message is received again at a later time, it will not be retransmitted.

Another contribution from Gnutella is that it implements the idea of decay. Each message has a TTL (time-to-live) number. As it passes from host to host, the TTL is decremented. When the TTL reaches 0, the request has lived long enough and is not retransmitted again. This explicitly prevents wasting network resources and keeps the resource tiny.

### 2.3.3 Distributed Computing

The distributed-computing solution divides an application into a huge number of essentially independent computations [Fox01] so that each participating node is ruled out of a number of separate work chunks. An example is SETI@Home [set].

SETI@Home is an Internet level distributed computing project that takes full advantage of P2P technology to exploit spare computing power. The client software only works when the computer is not used. It sets up the connection to the SETI@Home server to download its portion of the problem being solved; until the problem is solved, no further communication with the server is required [Wil01]. When the client software finishes the work, it re-connects to the server, sends back the result, and obtains a new task. In order to keep a work record, it writes the processing log into the disk. The primary P2P contribution of SETI@Home is to provide a mechanism for dividing a large task into multiple smaller tasks.

G2 Project [g2] provides programmers with a generic cycle stealing framework. One of its sub-projects is G2:P2P takes the advantages of P2P networks and implements file sharing, distributed com-
puting by integrating with .NET Remoting.

### 2.3.4 Other P2P Applications Examples

Core Technologies or services include P2P management, messaging, security, and client grouping, as well as file registration, discovery, and access capabilities. They define community standards that allow peers to join and leave the community. Jini [CDW00] [jin] and JXTA [jxt] are two important projects developed by Sun Microsystems. Jini has a simple model for dynamic self-defining objects and JXTA aims at core P2P capabilities, including peer grouping and security.

Usenet is the grandfather of today’s new peer-to-peer applications [Wil01] and was created in 1979 by two North Carolina graduate students. The Usenet system is based on the Unix-to-Unix copy protocol, which allows one UNIX machine to automatically dial another, exchange files with it and disconnect. This mechanism provides a way for two computers to exchange information like email, files, system patches, or other messages. Usenet today uses a TCP/IP-based protocol, which allows two machines to discover new groups efficiently and exchange new messages in each group.

The significant P2P contribution of Usenet is the means for machines to talk to each other, allowing messages to be posted and disseminated over a network. By providing a well-defined protocol, every machine is able to provide services. Usenet was the first true P2P application.

### 2.3.5 Summary

The above P2P applications can also be classified into two models: pure P2P networks and hybrid P2P networks [R01]. A pure P2P network is fully decentralized and the network is not affected if any peer is removed from it, whereas a hybrid P2P network requires a central server that provides parts of the offered network services [Mas02]. Ideally, P2P means a system without any central component. However, hybrid components are often imported as a means of adding more P2P functions to the application, such as distributing, caching and multiple networks interface.

Table 2.1 and 2.2 summarize P2P applications and P2P forms.

### 2.4 Related Technical Topics

P2P technology has been one of the hot topics in IT industry and research. The research in this area falls into the following topics:

- **Metadata.** Metadata is data about data. P2P applications allow the resources from one networked device to be available on-line. The provider must describe the resources so that they can be available on the Web and can be easily discovered. Metadata is the tool to describe the resources in each P2P application and to search resources on the network. Therefore, the research in Metadata involves:
### Table 2.1: Comparison of Current P2P Applications

<table>
<thead>
<tr>
<th></th>
<th>ICQ</th>
<th>Jabber</th>
<th>Napster</th>
<th>Gnutella</th>
<th>Past</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>hybridP2P</td>
<td>hybridP2P</td>
<td>hybridP2P</td>
<td>Pure P2P</td>
<td>P2P</td>
</tr>
<tr>
<td><strong>Instant Message(IM)</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>File Sharing</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Distribute Computing</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Across Platform</strong></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Aim</strong></td>
<td>To exchange instant message and to notify the user that his friend is online.</td>
<td>To develop a system, which allows the user to communicate with each other across various IM system.</td>
<td>To share MP3 files</td>
<td>To sharing files</td>
<td>To develop a storage management application.</td>
</tr>
<tr>
<td><strong>Typical Features</strong></td>
<td>Text exchange. A typical example of IM.</td>
<td>XML technology. It helps AOL to change its communication protocol.</td>
<td>Pointers stored on the server-indexing.</td>
<td>Broadcasting message mechanism; UUID; Server independent; Start points of P2P system are computers with static IP.</td>
<td>Efficient routing; Caching the file.</td>
</tr>
<tr>
<td><strong>Related to the Project</strong></td>
<td>Initiate the idea of sending messages between users in our system.</td>
<td>XML technology is applied through the whole project.</td>
<td>Attributes are used to describe the details of sharing files.</td>
<td>Broadcasting and UUID technology are applied in the system via the JXTA platform.</td>
<td>Caching is used in our framework for quick searching.</td>
</tr>
<tr>
<td>Chord</td>
<td>Intel Philanthropic</td>
<td>SETI@Home</td>
<td>Protégé</td>
<td>Proem</td>
<td>JXTA</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------</td>
<td>-----------</td>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>P2P</td>
<td>P2P</td>
<td>Not P2P</td>
<td>Not P2P</td>
<td>P2P</td>
<td>P2P</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Mobile devices</td>
<td>Yes</td>
</tr>
<tr>
<td>To develop a lookup service for a dynamic P2P system.</td>
<td>Recent P2P applications are suitable for distribute computing.</td>
<td>To develop a distributing system to efficiently use the computer resources.</td>
<td>To develop a system allows different software to interact.</td>
<td>Develop a P2P system for mobile collaboration.</td>
<td>To develop a protocol for different P2P applications.</td>
</tr>
<tr>
<td>This technology is suitable for a system with frequent nodes arrivals and departures. It uses the key mapping.</td>
<td>Distribute a typical task into small tasks, but still cannot provide a random sharing software technology.</td>
<td>Expect to fully take P2P technology, but it provides an idea for distributing computing.</td>
<td>Knowledge-based; Component based; Interaction between different software.</td>
<td>User profiles; This model is only for informal communication and exchange message.</td>
<td>Interoperability; Platform independence; Ubiquity Protocol based.</td>
</tr>
<tr>
<td>The framework uses peer-profile technology.</td>
<td>The project uses the idea of distributing computing for Remote Computing.</td>
<td>An example of distributing computing for Remote Computing.</td>
<td>Provide an idea of sharing different software and helpsto design the content of the peer profile.</td>
<td>Provides the reference of deciding the content, especially the content of peer name.</td>
<td>A basic platform. Pipes, communication protocols; Peer groups service and message passing, advertising service.</td>
</tr>
</tbody>
</table>

Table 2.2: Continued Comparison of Current P2P Applications
2.4 Related Technical Topics

- Designing ways of quickly searching for resources
- Standardising resource descriptions available to peer applications and end users, and
- Definition of common descriptive concepts for each P2P community.

- **Performance.** A P2P system is decentralized. Performance is an important factor for a P2P system because of the following three reasons:

  Firstly, a P2P connection is direct and depends on each peer’s personal resources. To support multiple simultaneous connections, the performance requires minimising traffic and balancing the load.

  Secondly, the communication between two peers can be crossed over some other peers. If one peer in the middle is unreachable, TCP/IP would take several minutes to time out the connection and the time to reach to the end peer would be the time of each TCP/IP time-out multiplied by the number of peers. To avoid this situation, it is necessary to reduce the number of peers that messages pass through.

  Thirdly, peer presentation and participation support P2P communities, particularly a P2P file-sharing community. If all peers in this community consumed resources without sharing resources, the community would not exist. Hence, the P2P system designers must have a method to deal with this problem.

- **Trust.** Trust plays an important role in P2P systems. Peers in these systems share not only files but also computing processing. The trust risk that each peer has to face is a result of the servers that the peers interact with, the size of shared files and the availability of updated files. For example, in the Napster system, the individual user installs the Napster client software and the software sends the list of MP3 files, related descriptions and the local machine IP address. In this case, the user must trust the Napster server, other peers, and the integrity of shared MP3 files if the user wants to copy these files into their local computer. If one peer updates the shared files, it must provide the updated files for others. On the other hand, the server also has to trust the peers that it connects to. Both sides take the trusting risk. To reduce the risk, a secure channel can be used to access to server and destination resources and to restrict the privilege of published files. Therefore, the trust principle of reducing risk is that the peer must reduce the number of peers in the community to be trusted.

- **Accountability.** One problem in Napster and Gnutella is the large consumption of bandwidth. Consequently, peers can abuse P2P systems by providing bad information, refusing to offer services when they are needed, and pretending to be other peers. To address these problems, the peer can limit the number of connections, limit data that is uploaded or downloaded, or limit the number of peers.
• Reputation. Because some P2P systems are not purely decentralised, they still require central servers in the middle to transfer data across different domains. These servers must operate in a distributed fashion and also act as a central server.

• Security. Security is a critical issue for networks, as well as for P2P applications. To protect shared resources from being abused, security technology must be used in a P2P system. The most common method is SSL. Since P2P communications are mutual, both peers who share resources must complete the invitation protocol (encrypted key installation) then these new pair keys need to be established for communication. Finally, both peers need to install a group key, in case other peers want to join the group. The shared data and communication messages are encrypted by the private key and can be opened by the public key. By using the cryptographic keys, the shared resources can be protected on each local peer machine.

• Interoperability through gateways. Most peers are on a local network, behind a firewall or on a different domain. To enable interaction between them, the communication messages must find their way over every connected network to their destinations. Consequently, the major problem is how each message finds its way to the peer in different networks. There are five solutions from different P2P applications.

The first solution is from Freenet [CSWH01], which uses the best path. If the information is in Freenet, the path length is short and it takes a short time for the message to reach the destination. Hence, the peer always tries to use the shorter path to pass the message.

The second solution is from Gnutella [gnu]. It limits the time-to-live and the maximum path radius. Within the Gnutella network, messages are broadcast to all peers and do not have to deal with the issue of choosing the right path.

The third solution is from Mojo Nation [moj01]. In Mojo Nation, there are content trackers that keep the content lists and the addresses of the peers, which can be retrieved at any time from these trackers. In order to integrate gatewayed trackers, a proxy peer is introduced in Mojo Nation. The content tracker is searched firstly. If it fails, the gateway will be searched and a proxy is used as well.

The forth solution is from Publius [WRC00]. The files in the Publius system are split into a number of parts and each part is sent to a different server. The length of each part’s path is 1. When the file is reconstructed, each part must be passed through a gateway. Therefore, most Publius nodes are gateways.

The fifth solution is from Free Haven [DFM01]. The files in Free Haven are divided into parts and each part of these files gets routinely traced. Consequently, each part can find a gateway.

In general, when a P2P application is designed, the above technical issues must be taken into consideration.
2.4.1 Multi-Agent Systems

Multi-Agent Systems (MAS) technology promises distributed operations, inherent executions and dividing the problem into small parts and conquering them by employing a number of robots via the internet. A MAS is defined as a loosely coupled network of problem solvers that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver [DL89]. These problem solvers are called agent and agents are heterogeneous and autonomous in nature [Syc98]. Katia P. Sycara defines the characteristics of MAS as the following:

- Each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint;
- There is no system global control;
- Data are decentralized;
- Computation is asynchronous.

Bond and Gasser [BG88] listed challenges of MAS and Katia [Syc98] added six more challenges. Gasser [Gas91] provided solutions for these challenges. The major challenge is that MAS is coherent collective behavior, which means two aspects: adaptation and collaboration.

Adaptation in MAS can be either agent level or system level [JZW03]. There is much research on agent adaptability at the agent level [BW01][DS97][FU98]. The technology used bases on machine learning or decision making [BGM00][BM01], many of them focus on centralised processes to formulate models or strategies [JZW03]. The adaptation at system level [JZW03] employs a type of dynamic role-filling mechanism to model the adaptation and meanwhile defines the agent interaction protocols as entities independent of roles and agents. Therefore, the join, withdrawal, or replacement of agents will not impact other agents and the agents do not need to adjust or change their behaviors or specifications to adapt to the changes of system structure and interaction protocols.

Agent in MASs acts as an active communicating entity and interacts with other agents or environment to provide services. Therefore, the agent consists of the following components [FG98]:

- Mental attitudes
- Behaviors Specifications
- Interface is used to interact with other agents
- Adaptation module allows agent to add more functions.

In order to collaborate with each other, the agent’s interface includes:

- services that the agent can provide
- requests to ask for services
channels for its communication with others, which are not only the media of services transmitting, but also the ports for sending/accepting messages.

The agent that provides the services acts as a server and the agent that requests services is a client. There is much research investigates in communication [FFMM94] [HS98] [PFL+98], which includes three areas [FM99]:

- A common agent communication language and protocol. There are two ways to design the communication language [FM99][Gen97]: one is to use programming languages such as Java [AG98] and the other one is to use declarative language such Knowledge Query and Manipulation Language (KQML) [Gen97][FTP+95][FFM92]. KQML is a language and protocol for exchanging information and knowledge between agents. It is also a message format and a message-handling protocol to support run-time knowledge sharing among agents [FFMM94] [FLM95].

- A common format for the content communication. It defines the specification schemes for describing the communication content such as Knowledge Interchange Format (KIF) [GF][OBC00][Gin91]. KIF is a computer-oriented language for the interchange of knowledge among disparate programs and has declarative semantics (i.e. the meaning of expressions in the representation can be understood without appeal to an interpreter for manipulating those expressions). It provides for the representation of knowledge, for the representation of non-monotonic reasoning rules and for the definition of objects, functions, and relations.

- A shared ontology. To support the sharing and reuse of formally represented knowledge among AI systems, it is useful to define the common vocabulary in which shared knowledge is represented. A specification of a representational vocabulary for a shared domain of discourse - definitions of classes, relations, functions, and other objects - is called ontology. Pragmatically, a common ontology defines the vocabulary with which queries and assertions are exchanged among agents. Ontological commitments are agreements to use the shared vocabulary in a coherent and consistent manner. The agents sharing a vocabulary need not share a knowledge base; each knows things the other does not, and an agent that commits to ontology is not required to answer all queries that can be formulated in the shared vocabulary. A commitment to a common ontology is a guarantee of consistency, but not completeness, with respect to queries and assertions using the vocabulary defined in the ontology [Gru93][WG97].

It is important that agents not only have ontology to conceptualise a domain, but also that they have ontology with similar constructions [FM99].

To support the collaboration between agents, research efforts focus on MAS architectures standardisation. These include the Object Manager Group (OMG), the Foundation for Physical Agents (FIPA), the Knowledge-able Agent-oriented System (KAoS) group, the General Magic group, and the Open Agent Architecture (OAA).
• OMG focuses on standardisation that is required to make products developed with agent technology work together [VOO95]. The OMG model outlines the characteristics of an agent environment composed of agents (i.e., components) and agencies (i.e., places) as entities collaborate using general patterns and policies of interaction [FM99].

• FIPA defines a series of specifications that promotes the interoperation of heterogeneous agents and the service that they can represent. These specifications can be categorised as agent communication, agent transport, agent management, abstract architecture and applications [fipa][fipb]. The framework developed by FIPA is described using a reference model (which specifies the normative environment within which agents exist and operate), and an agent platform (which specifies an infrastructure for the deployment and interaction of agent [FM99].

• KAoS architecture addresses the lack of semantics and extensibility of agent communication languages and provides an open agent communication meta-architecture in which any number of agent communication languages with their accompanying semantics could be accommodated. The key technology of KAoS is shared knowledge about message sequencing conventions (conversation policies) [BDC+95].

• The General Magic group explores the use of programs that can be shipped across a network to support the notion of traveling agents. This program segment includes local state to remote machines and receiving responses asynchronously [VOO95].

• OAA is a framework for integrating a community of heterogeneous software agents in a distributed environment [CM01][MCM99]. Agent in OAA society is defined as a software process and follows the OAA conventions. The agent must register the services that it can provide, must speak the Interagent Communication Language and share functionality common to all OAA agents.

• CARBA [JS99] is similar to CORBA and is built on the role-based in MASs.

MASs promise to be a valuable software engineering abstraction [FM99]. With machine independent programming languages this system can be widely adopted.

2.5 Conclusion

This chapter presents a description of P2P applications and highlights the key technology in typical P2P applications. It also reviews the related research in multi-agent systems. This chapter concludes with a summarised table of P2P technology, which provides guiding principles for the system design that is presented in the following chapters.
Chapter 3

Internet Robot

An Internet Robot is a robot that has the ability to connect to the Internet and also to be approached via the Internet. One of the significant advantages of such a robot is that it is not limited by physical distance constraints. A user can log on to the Internet anywhere around the world and control the robot.

This topic has recently been a hot topic in the robotics and control fields. The research aims to define a software framework that provides an efficient communication method to control an Internet Robot, and also to build a base for further research. This thesis is based on the study of networked robot control and available control approaches, which guides the framework design.

3.1 Overview

The chapter is organized as the follows:

- Section 3.2 surveys relevant research in the field of networked robot control.
- Section 3.3 covers the control modes of the Internet Robot and considers their relative issues.
- Section 3.4 presents a description of Behaviour-based and Event-based Robotics.
- Section 3.5 presents a summary of the chapter’s findings.

3.2 Networked Robots Controlling

A number of network-based robotic systems have been developed [DL98][GM95][uwa][CL97][VB96][BT98]. The controlling messages are transmitted via various media, such as radio, microwave and Internet. The most preferred method is the Internet, which enables control of robots from any geographic location. However, the detailed implementation and addressing of issues are different.

The system developed in the ART (Advanced Robotics and Teleoperation) Lab [LCM00] is used to control the robots via the Internet. In this system, communication is only between the operator and the robot and the communication protocol is client/server. There is a workstation deployed as a server, which
transfers the messages between the operator’s computer and the robot. Therefore, this research focuses on the problems that random time-delay and bandwidth constraints in relation to the traditional network protocol.

The Autonomous Navigation and Sensing Experiment Research (ANSER) project [usy] aims to develop Decentralised Data Fusion (DDF) [DWS01][EDW01][EDGW00][MNSH02] and Simultaneous Location and Mapping (SLAM) methods. In the DDF System, there is no central mode and sensor nodes connect with each other directly. The communication architecture of DDF is point-to-point, which does not provide services to each other.

Network-based robots with Internet control have been developed [LCM00][HKK01][TD97][SW95]. The communication mode is client/server with a HTTP interface. In this architecture, there are two participating parts: the controlling part, which is the operator, and the controlled part, which is the robot. The communications between them include commands, arguments and measurement results. The robots always wait for the commands from the operators and there is no direct interaction between robots. The research mainly focuses on the problems derived from the traditional network mode and control methods.

3.3 Robot Control Approaches

The control modes of the Internet Robot are divided into three types: direct control, supervisory control and learning control [LSST03]. The following sections describe these three types in detail.

3.3.1 Direct Control

In the direct control mode, communication between the operator and robots is direct via the Internet and other media such as radio waves. The operator sends the command and arguments that the command requires to the robot, then waits for the results from the robot. When the robot receives the command and its arguments value, it carries out the required tasks and sends the feedback to the operator after it has finished the task execution. When the operator gets the information from the robot, he or she can make further modifications to the command and the arguments, if necessary.

In this system, the operator must understand the characteristics and behaviour of the robot. The robot executes tasks under the direction of the operator. This mode can be applied in a situation where the robot performs a single task and waits for an order from the operator after having finished each task. This mode can also be used for the robot programmer to test and debug his codes. However, in real-time control, this mode suffers from the Internet’s high latency and low bandwidth.

In order to overcome the latency of the Internet, there are three solutions available:

- **Predicted Solution**

  The operator foresees or guesses some circumstance that will most likely happen, and then makes the robot’s behaviour automatic by creating a software packages to guide the robot in the virtual
3.3 Robot Control Approaches

<table>
<thead>
<tr>
<th>Solution Name</th>
<th>Definition</th>
<th>Focus</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted Solution</td>
<td>The operator predicts what will happen next and inputs the command and information required by the robot beforehand.</td>
<td>The prediction of the environment that robot are likely to encounter.</td>
<td>This solution is used in a situation where the information that the robot requires is not available.</td>
</tr>
<tr>
<td>Simulating and Planning Display Approach</td>
<td>The operator controls a robot in a simulated environment. This environment is modelled on the environment that the robot will encounter.</td>
<td>The development of a simulation and planning environment.</td>
<td>This solution is suitable for a situation where the operator can define the environment in which the robot will operate.</td>
</tr>
<tr>
<td>Wave Transformation shape</td>
<td>This approach is that the operator defines some reference parameters, since the control signals are continuous.</td>
<td>The algorithm of the parameter definition with respect to control signals.</td>
<td>The control signals are continuous and change like a regular wave.</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of three Control Solutions

environment. This approach is often applied in a situation where the robot cannot get valid information or values from the network, or whilst the processes in a robot are waiting for these values.

- **Simulating and Planning Display Approach**

  This approach is different from the Predicted Solution. It simulates a situation that the operator judges will happen. The operator controls the robot in a virtual environment. After each execution, the necessary data and command from the operator is stored and transferred to the robot. The commands and their arguments are always ahead of robot execution. Consequently, this approach can tolerate a network-transmitting delay.

- **Wave Transformation Shape**

  In a situation where the control signals are continuous and their Nyquist shape is similar to the wave transformation, the control signals can be predicted based on the wave transformation shape. This can also compensate for the transmission delay time.

Table 3.1 compares these three solutions mentioned above. In summary, the communication method between the operator and the robot is direct. The robot is totally under the control of the operator. There
are three potential solutions to the major problem of network delay. All research efforts focus on the compensation of losing time over the network.

### 3.3.2 Supervisory Control

In the supervisory control system, the robot has high capability and is intelligent. It is able to make some simple decisions and to operate automatically without directions from the operator. It will only need guidance from the operator when it is in uncertain circumstances.

In the supervisory control system, the robot receives tasks from the operator and executes the tasks automatically. In order to keep the robot running automatically, the operator must assign the robot high in-built capabilities and intelligence and consequently the requirement for the robots hardware must be high.

### 3.3.3 Learning Control

Learning control is an approach whereby the operator teaches the robot how to carry out a task in an environment. After having learnt a certain amount of skills, the robot can do the same job repeatedly without any interaction from the operator. Furthermore, it can handle some complex tasks based on the knowledge that it has achieved from the human operator.

When the robot repeats the same task that it has been taught, it is in an absolutely automatic status and can complete the task without any direction from the operator. Therefore, in this period there is no communication between the operator and the robot.

### 3.3.4 Summary

Robot control can be classified into three modes in terms of the communication method between the operator and the robot:

- The direct control approach allows the operator to guide his robot directly; consequently, it avoids uncertainty when the robot cannot make a decision. Because the operator interacts with the robot through the Internet, the robot does not have independent capabilities and communication will suffer from network delays.

- In contrast, a supervisory control environment lets the robot make decisions and utilize higher intelligence abilities. The robot needs direction from the operator only when it is in a confusing circumstance. This approach avoids some cases of network delay and also speeds up the performance of the robot. But it also causes some problems when the robot makes wrong decisions and it still suffers from network delays when the robot has to communicate with the operator.

- Learning control allows the robot to carry out some repeated tasks automatically. By using this approach, the robot can be in an automatic mode when it repeats the same task. However, this
approach is not suitable for unforeseen circumstances. It also requires the robot to judge whether it is in a situation that it has already learnt to deal with.

Table 3.2 compares three communicate mode, lists the relative research directions and summarizes their applications.

### 3.4 Event-based and Behaviour-based controls

There are two approaches to control an Internet robot [LSST03] and this section will describe these two methods in details.

#### 3.4.1 Event-based Control

Internet-based control is characterized by random time-delay and the time-delay variables are unstable. Therefore, it is impossible to design a control algorithm to model the maximum delay. Instead of designing a suitable control law, the event-based control method aims to deal with the random time-delay. It adopts a non-time based reference system in the control algorithm, which introduces an event-based action reference parameter that relates to the real-time sensor measurement and is independent of time. This method can successfully reduce the effects of time-delay constraints. By applying this method, the system keeps idling until the “event” disappears.

#### 3.4.2 Behaviour-based Control

The behaviour-based approach is a method of controlling mobile robots in a multi-robot domain. “Behaviour” in this concept is a process that the robot performs. The robot processes the data or measurement values that are input from sensors or other devices including robots and returns the results back to the requester.

In this approach, each process can be treated as an individual software package. These software packages can be composed at run-time to perform more complex tasks and can also be reusable. Complex behaviour is composed of a number of smaller processes, which are small individual software packages. Each robot system is a collection of these software packages.

The behaviour-based approach efficiently improves the performance of robots and reduces the uncertainty of robot systems, particularly in a multi-robot domain.

#### 3.4.3 Summary of event-based and behaviour-based controls

In this section, two popular robotic control approaches have been introduced: event-based control and behaviour-based control. Both of them focus on efficient control of robots and reducing uncertainty and real-time delays. Event-based control introduces a parameter that relates to real-time measurement and is independent of real-time.
## Direct Control

### Definition
The operator controls the robot directly.

### Applications
For situations when each task does not link to the next task.

### Focus
On developing solutions to avoid network delay and methods to compensate for the time lost during the data transmitting.

### Research Direction
To develop an efficient algorithm or method to compensate for time loss. However, real-time situations will not normally match planned situations.

### Advantages
Reduces the network delay.

### Disadvantages
Cannot be used in the situations that have high requirement for real-time parameters.

## Supervisory Control

### Definition
The robot makes decisions and only needs directions from the operator when it is in uncertain circumstances.

### Applications
For situations when the operator can clearly foresee future circumstances.

### Focus
On a methodology that teaches the robot and development of software packages for the robot and definition of automatic performance.

### Research Direction
To develop robot hardware and software packages for the robot so that it can operate automatically.

### Advantages
Allows the robot to run automatically without the operator direction.

### Disadvantages
Unsuitable for a complex situation where the robot cannot make decisions. Can also lead to some severity mistakes.

## Learning Control

### Definition
The robot learns some skills that it can perform automatically when it repeatedly does the same tasks.

### Applications
Suitable for situations that require the robot to do the same task repeatedly.

### Focus
On a method of teaching the robot, that forks into two steams: software solution and situation mapping.

### Research Direction
To design software packages and define a method of mapping situations to this design.

### Advantages
Allows the robot to be in totally automatic mode when it carries out some tasks repeatedly.

### Disadvantages
Not suited to an ever-changing environment.

<table>
<thead>
<tr>
<th></th>
<th>Direct Control</th>
<th>Supervisory Control</th>
<th>Learning Control</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition</strong></td>
<td>The operator controls the robot directly.</td>
<td>The robot makes decisions and only needs directions from the operator when it is in uncertain circumstances.</td>
<td>The robot learns some skills that it can perform automatically when it repeatedly does the same tasks.</td>
</tr>
<tr>
<td><strong>Applications</strong></td>
<td>For situations when each task does not link to the next task.</td>
<td>For situations when the operator can clearly foresee future circumstances.</td>
<td>Suitable for situations that require the robot to do the same task repeatedly.</td>
</tr>
<tr>
<td><strong>Focus</strong></td>
<td>On developing solutions to avoid network delay and methods to compensate for the time lost during the data transmitting.</td>
<td>On a methodology that teaches the robot and development of software packages for the robot and definition of automatic performance.</td>
<td>On a method of teaching the robot, that forks into two steams: software solution and situation mapping.</td>
</tr>
<tr>
<td><strong>Research Direction</strong></td>
<td>To develop an efficient algorithm or method to compensate for time loss. However, real-time situations will not normally match planned situations.</td>
<td>To develop robot hardware and software packages for the robot so that it can operate automatically.</td>
<td>To design software packages and define a method of mapping situations to this design.</td>
</tr>
<tr>
<td><strong>Advantages</strong></td>
<td>Reduces the network delay.</td>
<td>Allows the robot to run automatically without the operator direction.</td>
<td></td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td>Cannot be used in the situations that have high requirement for real-time parameters.</td>
<td>Unsuitable for a complex situation where the robot cannot make decisions. Can also lead to some severity mistakes.</td>
<td>Not suited to an ever-changing environment.</td>
</tr>
</tbody>
</table>
Behaviour-based control treats behaviour as an individual process unit. A complex behaviour is divided into a number of these units, which are software packages. These software packages can be combined together at real-time and can also be transported from one robot to another robot. Therefore, this method reduces the uncertainty of robotic systems.

3.5 Summary

This chapter surveys a number of projects wherein robots are controlled via the network. However, most projects are based on the traditional network architecture and hence their problem-solving methods are derived from this architecture. The ANSER project brings a new point-to-point system to share data directly, but in this system robots still cannot provide services to each other directly.

This Chapter has analysed three control modes. In real-time control, these three control modes can connect with each other and are used in most robotic system.

Also in this chapter, two common robotic control approaches have been described. They efficiently reduce the uncertainty of robotic systems. However, they cannot resolve the network problem.

Although the control approaches help to improve the performance of real-time collaboration, they still cannot fundamentally solve the following problems:

- Uncertain time-delay and bandwidth constraints;
- Uncertain data-loss;
- Data transmission security.

In order to address these problems, the framework not only uses the proper control approach, but also applies the right communication protocol.
Chapter 4

Robot Colony based on the JXTA platform

4.1 Overview

This chapter describes the design and implementation of the framework Robot Colony, which is built on the JXTA platform. It also investigates the JXTA technology and draws the architecture which integrates its own functions with the JXTA platform. Then it describes in detail the framework implementations on the JXTA platform. Finally it analyses the framework for future work needed be done.

4.2 Introduction

Since the late 1990s, commercial companies, research organisations and institutions, have been developing P2P applications dramatically such as Jabber [jab], ICQ [icq], Gnutella[gnu], and so on. JXTA [jxt] is one of these projects and aims to develop P2P core technology, which includes peer grouping, discovery and file registration, by defining P2P protocols. It can allow any further P2P applications based on the JXTA Platform. As the capabilities of wireless communication in embedded devices are increasing, there is a tendency to apply the P2P technology on wireless embedded devices.

The Robot Colony will be built on this platform and help robots including the wireless robot to form a cooperative system. The system is highly dynamic. In order to discover each other and available services, a customized profile-based mechanism [Kor01] is applied to the framework. The Profile-based mechanism in this sense is actually an XML description protocol to define each robot so that it can communicate with other robots. A profile must be embedded inside a JXTA message since the basic communication tool of the framework is the JXTA Pipe. To allow interaction, the framework not only supports remote control, but also allows the robot to dynamically load classes from other robots. The following section will describe the design goals of the framework based on the JXTA platform.
4.3 Design Goals

The Robot Colony is built on the JXTA platform, which is already provided with the basic P2P communication, is able to help robots communicate with each other directly and to form a P2P group. However, the purpose of this robot group is different from other P2P groups in the JXTA applications because the robots group is driven by applications and the framework running on each robot must allow it to be controlled/monitored remotely by others. Therefore, the general design goals of the Robot Colony are to provide necessary functions to support robots interaction, to help robots to form a dynamic decentralised group, to enable load classes dynamically, and to enable to be invoked remotely. To achieve these general goals, the framework running on each robot must provide the following operations:

- **Advertising**
  A robotic system is highly dynamic and each robot joins or leaves the community frequently. In order to monitor and cooperate with other robots, a robot must know the states of the other robots. This operation is used by a robot to broadcast its information, the status of devices and available services to the community members when it joins the community and to notify the members when it leaves.

- **Data Transfer**
  One advantage of the P2P technology is data sharing and the data can be a text file, a binary file, a command line message or an instant message. In a robotic system, for example, Robot A needs software, which is not in its system but is available on other robots. Therefore, the Data Transfer operation allows the robot with the software to send the code and its description to the demanding robot. To support the collaboration between robots, the operation also provides the tools for sending instant messages, commands, variable values and the execution result.
  In order to improve the robotic system performance, this operation can also be used to transfer a file like an image file to a powerful PC so that the PC does the image processing and the robot uses its resources for other purposes. When the PC completes the processing, it uses this operation to send the result back to the robot.

- **Dynamic Class Loading**
  Once a robot gets the binary software, the robot saves the software into its system. After having stored the software, the software is like other softwares having been installed on the robot’s system and the robot can invoke it whenever it needs. This operation is one that allows a robot to load software into its system and the software is ready to be invoked by itself or other robots.

- **Remote Execution**
  In order to construct a robotic system, one basic condition is that these robots collaborate with and control each other. This operation provides a method for a robot to require other robots to execute some functions on their systems and to get the results back.
• **GUI - User Interface** is a graphic interface that allows the user to control and update the robots and to debug the code that installs in the robot. It also helps the user to initiate the state of the robot, such as which services the robot can provide. This GUI is an optional function and can be activated whenever the user needs.

In summary, the objective of this project is to design a framework with the necessary functions so that each robot can cooperate with each other to achieve the common goal. The above functions must be integrated with the JXTA platform so that the framework can use the JXTA P2P communication tool. The framework can also combine these functions together to suit for the application requirement. To be able to implement the design, it is necessary to analyse the base platform. The next section will describe the JXTA Platform.

### 4.4 Basic Platform

Project JXTA sponsored by Sun Microsystems follows Jini [jin]. It is language independent and different from the other P2P applications with a common platform so that developers can meet their special requirements based on the JXTA technology. By using this technology users can establish a communication connection across different platforms. In this project, the JXTA platform is selected as the basic platform for the following reasons:

- It is developed on a small number of protocols and these protocols are independent.
- It is neutral of programming languages.
- It is independent of transport protocol.
- It is implemented and integrated by heterogeneous devices.
- It allows developers to build different services.

#### 4.4.1 The JXTA Structure

The JXTA architecture consists of three layers: platform, service and applications (Figure 4.1). The platform layer, which is called Core layer, comprises of Peer Groups, Peer Pipes, Peer Monitoring and Security Primitives so that all peers can interact with each other. This layer also contains Peer IDs and is the core element in the JXTA architecture. It handles common P2P networking, transport and firewall. It is also responsible for creating peers and peer groups.

The services layer expands the capabilities of the platform layer and provides facilities like searching, indexing, discovery and membership. The peer services can create secure tools and the specific tool policies determined by the users. Searching, indexing and discovery are deemed as network services, but these services are not P2P network operation services so that they can be loaded as required.
The peer applications layer include instant P2P messaging, mail, file sharing and so on. It allows extended application functions developed on this layer and the applications can be viewed as a service in the platform. Therefore, there is no strict boundary between the services and applications layers. One P2P application can be a service for another P2P application. Services and applications are selected according to the need.

### 4.4.2 Overview of the JXTA Shell

The JXTA Shell sits between the JXTA application and the JXTA Platform and provides a command line interface between users and the JXTA core layer. The users can interact with the JXTA platform and the JXTA core layer through Shell Command line interface.

There is a set of commands in the JXTA Shell. Each command consists of a command name, followed by command options and arguments. These commands binary code are stored in the directories, which have the same name as the commands. When they are invoked, the commands would be dynamically loaded into the Shell.

Currently there are two kinds of shells in the JXTA platform: normal shell and SQL shell. Each shell can fork child shells by using command “Shell -s”. The normal Shell is used for normal command execution, while the SQL Shell is used as an interface to invoke the commands linking with the database.

The JXTA Shell implements the JXTA command interface like other JXTA commands. Therefore, it can be accessed programmatically by calling the JXTA command interface. This feature is very useful for any developed application because the developers can develop required functions that can access the JXTA core layer via the JXTA Shell object.

### 4.4.3 JXTA protocols

JXTA is divided into six protocols and these six protocols are independent, which are:

- **Peer Discovery Protocol.** This protocol provides the mechanism by which each peer describes
and can advertise its resources and also locates the resources from other peers. The advertisements are XML documents.

- **Peer Resolver Protocol.** The design idea of the JXTA is to allow each peer to send, respond and query, therefore, the peers implemented the JXTA protocols must have a common blueprint. This protocol is designed for this purpose. Once a peer discovers another peer in the network, it can send a message to that peer, but this does not mean that other peers will respond to that message.

- **Peer Information Protocol.** The JXTA allows peers to monitor each other and this protocol provides the method to get the status information about other peers.

- **Peer Membership Protocol.** According to this protocol, each peer can form a self-organized group and can also join in other group. This protocol provides the mechanism by which peers can sign a contract to join and leave the peer groups.

- **Pipe Binding Protocol.** This protocol provides the communication bridge between peers. If one peer wants to send a message to other peers, it must use this mechanism to build the endpoint connection so that the message can pass through this communication channel and reach other peers. This protocol is the fundamental tool for peer communication.

- **Peer Endpoint Protocol.** Suppose, peer A wants to send a message to peer B and they cannot communicate directly. Hence, peer A needs intermediary peers so that peer A sends the message to those peers and they pass the message down to peer B. This protocol is designed for this purpose so that the peer can find the routes for communication to remote peers.

The end-users can choose the above protocols depending on their actual usage.

The JXTA protocols have the following characteristics:

- The JXTA protocols do not define which programming languages implement them, therefore, devices written by different programming languages can interact with the JXTA protocols.

- The JXTA protocols do not require a specific operating system and hence are easy to implement.

- The JXTA protocols are independent of any network transport, and any security model.

With these protocols, peers can:

- Discover each other;

- Form a self-organised peer group and allow other peers to join the group;

- Advertise and discover network resources;

- Communicate and monitor each other.
4.4.4 JXTA Pipes

The basic communication tool on the JXTA platform is Pipes, which are virtual connection channels and link to peers. This communication can be either between peers or between processes in one peer. The functions of the pipes are divided into two groups. One is to link the sending and receiving messages between services or applications within a peer and the other is to provide the basic communication bridge between the ends of peers.

There are two kinds of pipes in a peer: output pipes, which send messages, and input pipes, which receive messages. These pipes can be re-directed.

EndPoint pipes dynamically connect to the peer endpoint at runtime via the Pipe Binding Protocol. These pipes link two peer endpoints together so that messages can be transmitted from one machine to another. In order to secure the information in the pipes, the end-to-end pipes can be enhanced into secure pipes. The endpoint pipes can also build the communication between one output pipe and multi-input pipes to propagate the messages.

4.4.5 JXTA Message Passing

Messages are the basic communication units and are like objects transmitted between pipes of peers. The JXTA protocols define the messages and each message must have an envelope, which indicates the receiver’s address (Peer name or Peer ID), a body and sender’s address according to JXTA Peer Endpoint Protocol.

The content of messages can be arbitrary, but messages are composed of a set of name/value pairs. The messages can be represented into two ways: text string and binary and these two representations are encoded in the body of an XML message following the standard scheme. Hence, the messages can be transferred to each peer crossing different networks and the services embedded inside the messages can also be propagated amongst peers by using the XML technology.

A message is also used to advertise peers, peer groups and pipes, services. In this case, a message is represented hierarchically as an XML document composing arranged metadata units and each unit is a name-value string. An advertisement message can be defined with lifetime for deleting expired resources and can also be republished.

4.4.6 Peers and Peer Groups

A peer is an individual unit in the P2P network and can be either a PC, a PDA, a server or a device. But it must implement one or more JXTA protocols. Each peer has a unique identified ID, which is generated at the beginning when the platform is configured, can advertise its resources and can also get services from other peers.

A peer group is a collection of peers that follow a common rule of sharing services, which is called policy. A peer can form a group and also can publish, discover, join and leave other peer groups. The peers in the same group can monitor each other and can also share the resources according to the peer
group’s policy. Each peer group identified by a unique ID has its own services and the peers in the group can obtain the services. These services include advertisement, discovery, and membership services. If a peer wants to get the services of a specific group, it must sign the contract and join that group. It does not limit that peer joining other groups. Hence, the peers in a particular peer group must have some common interests and they probably also belong to other groups.

4.5 Robot Colony Design and Implementation

The Robot Colony supports dynamic class loading and remote invocation by providing with some necessary general functionality [FSJ99]. It starts with listing requirements that collects some virtual examples explained in Section 1.1. The first version of the framework is the implementation of the examples and is a white box framework. The chapter does not cover the management and further development of the framework.

4.5.1 Functional Requirements

This section presents the functional requirements of the Robot Colony. The key part of this stage is to specify the behavior of the framework by giving a simple model of cooperative robotic systems.

The robots in a cooperative system always cooperate with each other to finish a specific task. Robot A wants Robot B to execute a command and to send the execution confirmation back. The situation of Robot B is that it does not have the command software, therefore Robot A sends the command class file to Robot B and Robot B loads it into its system after having got the whole file. Then Robot A sends a command line to Robot B and invokes the command remotely. After having executed the command line, Robot B sends the execution confirmation back to Robot A.

In this system model, the following questions need to be addressed:

- How does Robot A learn the situation of Robot B? Robot A and Robot B are in the cooperative robotic system. They must know the status of each other at the beginning when the system is functioning so that they can use available resources.

- How does Robot B get the command software and load the software into its system? Suppose that Robot B has got the software from Robot A, it must have the ability to load it into its system.

- How does Robot A invoke the command on Robot B? Up to now Robot B has loaded the software into its system, Robot A sends a command to Robot B and invokes that command on Robot B so that Robot B executes the command on its local system.

- How does Robot B send the execution confirmation back? When Robot B finishes the execution, it sends the result back to Robot A.
The messages circulated among the controlling robot and the controlled robot contains the command, the software and data. However, in order for successful communications to occur, both robots must agree to a common format for the messages so that they can be transferred and interpreted properly on both sides. Because the Robot Colony uses the JXTA platform as the communication channel, the messages must be in the JXTA messages format and follow the JXTA protocols. The following section describes the JXTA protocols that will be used for the Robot Colony.

4.5.2 Required JXTA Protocols

The framework of the project is based on the JXTA platform. The JXTA platform is designed for the communication between peers inside or outside network and it is composed of six protocols (Details see Section 4.4.3). The peers in the system are behind the firewall and in the same network domain, therefore, the communication is direct. Five of the JXTA protocols are used and these five protocols are Peer Discovery Protocol, Peer Resolver Protocol, Peer Information Protocol, Peer Membership Protocol and Pipe Binding Protocol. Their functions are the following:

- **Peer Discovery Protocol** is used to describe the robot and its services for other robots. In order to locate the available services a robot provides, this protocol defines the format to describe the services location and required arguments.

- **Peer Resolver Protocol** is used for bi-direction communication between robots. This protocol defines the communication blueprint in terms of sending, responding and querying that robots on both communication sides must follow. This is an essential element for building communications. Therefore, it is required to implement by the JXTA platform.

- **Peer Information Protocol** is used to monitor the status of a robot, which indicates whether the robot is on-line and whether its services are available.

- **Peer Membership Protocol** is used to define the group. As required by the JXTA platform, the robot automatically becomes a global group when the JXTA platform runs. The robots join together driven by the requirements of a particular application and trust in each other. Therefore, robots share resources and services without security limitation. In this case, Robot Colony uses the default global group that the JXTA Platform provides and follows its definition.

- **Pipe Binding Protocol** is used to connect two robots via pipes. Pipes are the basic communication tools in the JXTA Platform. This protocol is used to define the communication method to connect the JXTA pipes inner processes or outside processes. The four software components provided by the Robot Colony are application functions for the JXTA Platform and use the default JXTA input and output pipes. Therefore, these four functions must follow this protocol and interact with other processes locally or remotely.
This section describes the necessary JXTA protocols that the Robot Colony must follow to support collaboration between robots and be able to integrate with the JXTA platform. The next section will describe the architecture of the Robot Colony integrated with the JXTA Platform and also the location of these four components that the Robot Colony provides in the architecture.

### 4.5.3 Architecture

Section 4.3 describes four operations provided by the Robot Colony to support the collaboration between robots and these four operations are extended application functions to the JXTA Application Layer. They inherit the JXTA Application interface and are able to use all facilities provided by the JXTA Platform at the run-time. In order to utilise functions from the JXTA Core Layer, these four functions must interact with the JXTA Shell and call the interface at the programming stage. Like other functions in the JXTA Application Layer, these four functions can interact with the JXTA Service Layer directly, locate in the commands directories and are called/loaded dynamically. The GUI is an optional function and is used as a debugging and testing interface for the developers.

The framework architecture, illustrated in Figure 4.2, is divided into three layers:

- The lowest layer is the JXTA Core Layer, which provides the communication tool.
- The middle layer is the functional layer, which consists of the JXTA Application Layer, Services Layer and the JXTA Shell. The Robot Colony provides the following functions:

  1. **Advertising**
  2. **Data Transfer**
  3. **Dynamic Class Loading**

![Figure 4.2: Advertising Procedure via GUI](image)
4. Remote Execution

These four functions locate between the JXTA Application Layer and the JXTA Services and implement the JXTA command interface. Therefore, the user can use these four functions by calling the component name through the JXTA Shell.

- The top layer is a user-friendly interface and is responsible for the communication between users and the framework. It checks the commands directory and can enable the user to input the JXTA commands, execute the commands and get the execution results. It also allows the user to configure, update and define a description message for a robot. This message called the robot profile contains all necessary information in the format following to the JXTA protocols to make P2P communication and remote execution occur.

This layer is an optional layer, which means the user can activate when the user needs. The reason for it is because most robots do not have X windows and some users prefer to control their robots via command lines.

These three layers are designed as to be three independent components so that any of them can be replaced without affecting the rest. In order to enhance the framework functionality of the framework, developers can add more functions that implement the command interface and put the compiled source code into the JXTA command directory. The user can dynamically execute the new command by calling the command name without stopping the framework. The following sections describe the implementation of these four functions and how these four functions fit inside the JXTA platform.

4.5.4 Advertising

Advertising is an operation to construct a robot information message from either a configuration file or GUI - User Interface (Details see Section 4.5.9) and to pass the message onto other robots so that they can be notified of its existence, its available services and its devices status. This robot information message also called the peer-profile is saved in /etc of the platform. The message is an XML file describing the peer for other peers in the group and contains the peer name or ID, available service’s name and its attributes, and shared file’s name and its attributes.

Even though JXTA’s Peer Discover Protocol provides a mechanism for a peer to advertise itself by using the ID of peer, pipe and peer group, it does not provide the peer a tool how to publish its available files and services. The robots in the project interact with each other, so not only the ID of each robot is required, but also the services and files (source code) that each robot can provide.

The framework uses a peer-profile to advertise a robot so that the rest of the peers can get its information. The profile is packed into a JXTA message (an XML format file) and the JXTA Shell pushes the message into the JXTA pipe to broadcast to every peer on the network. Advertising still uses the JXTA’s Peer Discovery Protocol advertising mechanism, but the content of the advertisement has been expanded differently to the original one. The whole procedure is shown in Figure 4.3.
Once a peer gets the message, Advertising unpacks the message into a profile and stores it into its local system for future reference. This profile is modified whenever it needs.

In general, the function of this operation is to wrap the text string into a profile, unpack the message and store the data into the local system. In order to improve the speed of searching, the ideal way is to save the profile into the memory-cache and update the cache if a peer leaves the community.

### 4.5.5 Data Transfer

The framework is designed for the communication between peers. The communication content varies and the framework can be used for Internet chatting, for remote execution or for exchanging files. Therefore, a function - Data Transfer is added to the framework to support the robots collaboration communication. In order to distinguish the format of the data, XML metadata technology is used to pack different types of data like the one used in Jabber[jab].

Sharing data is one typical advantage of the P2P computing model. In the framework, the data is divided into three streams from the user’s perspective.

- **Instant Message**. Under a P2P environment, peers can communicate with each other and the instant message is the communication content between peers. In the project, the instant message can be a service name and its required variables. Like in UNIX, JXTA Shell commands are designed to execute on a local machine. This kind of data is the command line string and parameter values through the Internet sent to the executing peer to realise the purpose of the remote execution described in the next section, which is embedded inside an instant message.

- **Objects**. In order to update the peer system, it is necessary for the framework to have a function, which is to export the objects to other peers.
• **Files.** Some robots in the group are equipped with cameras. Image processing consumes high computing resources. Ideally the camera on the robot takes pictures and the robot sends these images to PC for processing. Hence, the framework provides a function to transfer different format files to other peers or a work station.

The JXTA Pipes support the transfer of any object, including binary code, data strings, and Java technology-based objects [Suna]. **Data Transfer** uses the available JXTA pipes. Considering that the data could be lost during the transition, **Data Transfer** combines JXTA pipes with XML technology to overcome this shortcoming. When the destination robot gets a file, it firstly checks whether the size of the file is correct. **Data Transfer** also defines the destination position of the transferred file so that the destination peer can save the file into the right position.

The difference between JXTA methods and **Data Transfer** is that the JXTA application simply uses the pipes to import and export the file without having a function of checking the file size and using the XML technology, which leads to a problem of losing data during the pipes connection. The JXTA platform does not have a method to tell the imported peer where it should save the file. **Data Transfer** is based on the JXTA importing and exporting files via pipes, but adds more functions to it.

Once it is possible to transfer a Java class file, which is a binary file, the executing peer and remote peer can dynamically load the class file into its platform. The next section describes **Dynamic Class Loading**.

### 4.5.6 Dynamic Class Loading

**Dynamic Class Loading** is an operation to dynamically load a class file, which is a Java class file in the project, into the system. When a robot needs to update its system, it downloads the source code from other robots. In order to run the software, **Dynamic Class Loading** must check whether the old version method is loaded and waits its stop before loading the updated classes.

The JXTA Platform uses the pipe mechanism to build the communicating bridge between peers. The shared **Data Transfer** in the framework loads the shared data into the right position, which consists of two steps. The first step is to check the size of the file. When a peer exports its file, it sends the file size and the location with the file. Therefore, the receiving peer obtains not only the file, and also the file size and its location. If the size of the file were not equal to the value in tag of “size” in the message, the framework would ask the sending peer to re-send the data package again until the size of the file meets the requirement. The second step is to load the object class into its system.

There are two ways to load classes dynamically - java.lang.Class(J2SE 1.4) or to create a custom ClassLoader inherited from java.lang.ClassLoader. If the programmer knows the class name and the behavior of that object at the writing code stage, then the first method is suitable. But it is required to define the behavior of the object and to load it dynamically: the difficult part is that the new commands must interact with the existing JXTA Platform. Therefore, an interface (called a contract for future commands) is created to define the behavior of the object, like the available JXTA commands. When the
user needs to load the command classes dynamically, the user calls the command name (the class name) to load the command into the JXTA Platform. Listing 4.1 is an example code of dynamic class loading.

Listing 4.1: Customized Class Loading and Creating an Instance

```java
try {
    // get the class name
    ...
    /* appClassName is the argument from the user input */
    // find the class and create an instance
    Class appClass = Class.forName(appClassName);
    app = (ShellApp)appClass.newInstance();
}
```

Once the JXTA Platform gets the command object, the next task is to get the JXTA Platform environment and to ask it to execute the command object. The next section describes how to execute the command remotely.

### 4.5.7 Remote Execution

Remote Execution is a basic operation in a robotic system with the aim of providing a tool that all devices can interact with each other. It allows a device to remotely control another device via the network. To achieve this goal, there are three elements required.

- Protocol Support: the actual transport mechanism for the devices communication.
- Interaction Support: the virtual interaction between the JXTA Cores of two devices.
- Communication Support: the requirement that defines the protocol between processes in a device.

In providing the protocol communication support, the type of P2P communication needs to be decided because the JXTA platform has already provided the communication protocols. In the project, pure direct P2P protocol is chosen and Instant Message protocol is used. Both technologies meet the requirement of remote execution, which is being direct and less time consuming.

The key to building an interaction between the JXTA Cores of two devices is the acquisition of an appropriate communication language. JXTA Shell permits interactive access to the JXTA Core via a simple command line interface [Suna]. The common communication language therefore is JXTA Shell command line, which is a text string.

The pipes are virtual communication tools used not only between devices, but also between processes. All communications inside the JXTA platform are based on the pipes. In order to keep the integration of the JXTA platform, the added applications connect with the JXTA core via pipes. For the convenience of further development, the communication between the added operations is also through pipes. In the future, if any new application uses any added operation, it simply links with the input pipe of that operation.
After having made the decision about the above three elements, a close look at **Remote Execution** procedure illustrated in Figure 4.4 and Figure 4.5 is described in the next section.
Remote Execution is an operation accomplished by two peers interacting with each other. The peer, which sends the command to another peer, is called Controlling Peer. The peer, which executes the command, is called Executing Peer. The Controlling Peer has to send the command line to the Executing Peer, illuminated by steps 1 to 4 in Figure 4.4. When it gets the result from the Executing robot, it needs to follow steps 5 to 7 in Figure 4.4 to display the result to the user.

In Figure 4.4, GUI - User Interface (See Section 4.5.9) gets the command input from the user and passes it onto Data Transfer via an inner pipe (Step 1). Data Transfer checks whether the command name exists in its local peer profiles (Step 2), if there is no such command, it will send the error message back to GUI - User Interface to notify the user. After having confirmed the existence of the command, Data Transfer constructs a data package in the JXTA Message format (Step 3). Listing 4.2 is an example of the command message. Data Transfer links the end of local pipe to the JXTA stdin and pushes the command message into the JXTA Core. At this time, the JXTA Core has built the connection with the executing robot (Step 4).

The Controlling Peer Core receives the message from the Executing Peer through the receiving pipe (Step 5); it then forwards the message to Data Transfer via stdout (Step 6). Data Transfer unpacks the message and extracts the result for GUI - User Interface to display to the user (Step 7).

The Executing Peer is responsible for the command execution. Figure 4.5 illustrates its six steps.

1. The Executing Peer’s Core gets the message from the Controlling Peers via the receiving pipe.

2. It redirects the message to Remote Execution through stdout to extract the command line and it meanwhile keeps the ID of the Controlling Peer.
3. **Remote Execution** asks the JXTA Core to execute the command by connecting the local pipe and the JXTA stdin. The difficulty of this part is how to let JXTA Core to execute the command. The key solution is to find the part of the JXTA Core, which executes the command, and to insert the command string into that part (See Listing 4.3).

```
// get the shell object
try {
    Shell shell = (Shell)(super.getClass().newInstance());
    shell.startApp(command);
} catch(Exception e) {
    println("Cannot run the shell.");
}

// get SQL shell
......

// deal with the Hashtable and HistoryQueue
......

// get the shell environment
ShellEnv env = getEnv();

// get the root shell process
ShellObject shellObject = env.get("SHELL");

// create the child shell
try {
    Shell sh = null;
    sh = (Shell)obj.getObject();
    // run the command
    runCommand(commandName);
}
```

4. Once the Core has finished the execution, it sends the result and the sending peer’s ID through the stdout to **Data Transfer**.

5. **Data Transfer** wraps the result into a JXTA Message back to the stdin of the Core.

6. The Executing Peer Core transmits the message back to the Controlling Peer’s Core via the sending pipe.

Now the whole **Remote Execution** procedure has completed, but sometimes users use piped commands,
which there are more than one command and the output of the front command is the input of the next command. The procedure is the same as above, except the Remote Execution on the remote peer extracts the piped command string and inserts another part of the JXTA Core, which executes piped commands.

Remote Execution is the main characteristic of the framework and provides the basic tool for the interaction between peers. It also offers the potential method for interaction between different software (which is out of the range of the project).

### 4.5.9 GUI - User Interface

The top layer of the framework is the user graphic interface (GUI), which allows the user to access the JXTA core and is compatible with the JXTA platform, as shown in the following figure 4.6. The aim of this part is to create a graphic window and the following functions:

- It gets the input from the user and constructs it into a peer profile for the Communication Protocol and Data Transfer.

- It displays the replied message and the result from Remote Execution.

The content of the profile, which describes the peer, is the most critical part. In order to construct the profile, the content of the profile must include the following two pieces of information:

- The peer information must contain peer’s name, available service names and file names. (In the project, the service means JXTA Shell commands. The file can be either a binary file or a text file.)

- Attributes information must describe the pre-condition and post-condition of each service, the location and size of the file.
In this GUI frame, there are three buttons for the user to input a peer name and to decide which services the user provides and which file is available to be shared. In order to describe the service and file, the user is required to input the attributes information for the respective file and service. When the GUI gets all inputs from the user, it will construct a profile, which contains three text strings, and send it to the Communication protocol.

The GUI frame has a text area to display the results from Remote Execution. It also allows the user to input the command for Remote Execution, like a JXTA Shell console. As what mentioned before, the GUI is an optional tool for the users and can be activated as needed. When the user starts this function, the GUI object connects with the JXTA input and output pipes, in which the user’s input is sent to the JXTA input pipe and the output from the pipe displays on the GUI. The profile that the GUI generates is exactly the same as the profile saved in the /etc directory of the JXTA Platform. The user can create and modify this profile instead of through the GUI. If the GUI makes any changes, the profile that it generates will overwrite the existing one.

In summary, the following table is the overall view of the contribution of the project. Note that the above five operations integrate with the JXTA Platform to form the whole framework in the project and they interact with each other.
<table>
<thead>
<tr>
<th>Module</th>
<th>Objectives</th>
<th>Problems</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advertising</td>
<td>To construct a profile into a JXTA Message in XML format; To send it to JXTA core; To get the message from JXTA core; To unpack it into a profile; To store it into cache.</td>
<td>To decide the message format; To make a JXTA compatible message; To send the message into JXTA core; To unpack the message; To store and to update the profile into cache.</td>
<td>To use XML technology; To understand the JXTA message structure; To use JXTA inter pipes; To reference the relative program how to save data into cache and how update the content in the cache.</td>
</tr>
<tr>
<td>Data Transfer</td>
<td>To construct file, instant message and command line into a JXTA message; To send it to JXTA core via the sending pipe; To get the package; To get the individual data pack; To direct them to the respective process.</td>
<td>To decide the format of message; To make JXTA compatible message; To send the message into JXTA sending pipe; To unpack the message; To direct the small data package to different processes.</td>
<td>To use XML technology; To understand JXTA message structure; To use JXTA inter pipes and sending, receiving pipes; To serialise the binary file into an array and assign the array size to the attribute of the file.</td>
</tr>
<tr>
<td>Dynamic Class Loading</td>
<td>To get the file and file’s attributes from Data Transfer; To load it into the peer system, which is the JXTA platform.</td>
<td>To make the decision between two classes. Loading technologies; Compatibility with current environment.</td>
<td>To analyse the framework functionality; Referring to Java dynamically loading class technology; To study JXTA Core technology.</td>
</tr>
<tr>
<td>Remote Execution</td>
<td>To get the command from Data Transfer; To send it to JXTA Core; To ask JXTA Core to run the command; To ask Core to send the result back.</td>
<td>To check whether command is available; To send it to JXTA Core via a pipe; To access JXTA Core; To find out which part executes the command; To link the pipes between JXTA Core and sending end.</td>
<td>To create a class to list available command lines; To send the command to JXTA Core; To read the source code related to JXTA Core and JXTA Pipes.</td>
</tr>
<tr>
<td>GUI - User Interface</td>
<td>To get the input from the user; To construct the input into a profile for the Communication Protocol; To display the result from Data Transfer.</td>
<td>To design the content that user needs to input; Create a GUI compatible for the JXTA platform.</td>
<td>GUI is written in Java; Reference to related profile-based technology; JXTA programmer’s &amp; guide.</td>
</tr>
</tbody>
</table>

Table 4.1: Framework Functions Implementation
<table>
<thead>
<tr>
<th>Related works</th>
<th>Characteristics</th>
<th>Result</th>
<th>Significant Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jabber, XML technology, PAST, Gnutella’s broadcasting technology.</td>
<td>JXTA compatible message; Offering quick searching profiles list</td>
<td>Have finished the packing of profile, sending it to JXTA core and unpacking the message.</td>
<td>Store the profile into the cache.</td>
</tr>
<tr>
<td>Jabber, XML technology, Napster</td>
<td>Using the attributes to describe the file’s size, location. Using the attributes to describe the command’s function, pre- and post-condition.</td>
<td>Have finished individual sending data.</td>
<td>Put all individual data package into one complete data package.</td>
</tr>
<tr>
<td>Java loading class technology, JXTA core technology.</td>
<td>Load the class dynamically with the function of checking the size of class. Currently JXTA has not this checking function.</td>
<td>Completed this task.</td>
<td>No requirement.</td>
</tr>
<tr>
<td>Unix Shell command and JXTA programming guide, Shell structure.</td>
<td>Check whether the command available in the system and execute locally, sending the result back. Currently no P2P application can allow the peer to execute the software remotely.</td>
<td>Totally meet the objectives, include single command line and piped commands. The excellent part is that it can pass the command line further.</td>
<td>No requirement.</td>
</tr>
<tr>
<td>Protégé, Proem, Napster</td>
<td>Our peer-profile includes software attributes and peer information, Protégé contains software attributes; Proem’s profile includes the user information.</td>
<td>We have finished design of profile’s content, GUI written Java, but the GUI is incompatible with JXTA platform.</td>
<td>Make the GUI compatible with JXTA platform.</td>
</tr>
</tbody>
</table>

Table 4.2: Framework Related Works
Chapter 5

Enhanced Robot Colony

The previous chapter described the design and implementation of the Robot Colony based on the JXTA platform. The Robot Colony used the P2P communication function from the platform and also implemented its own functions to support the collaboration between robots. These functions inherit from the JXTA customized interface and are embedded inside the JXTA platform. This chapter will analyse the Robot Colony and draw a conclusion of whether the JXTA platform is suitable for the robotic cooperation. It will also provide an enhancement for the Robot Colony based the analysis of the previous implementation.

5.1 Overview

In this chapter, the JXTA platform is analysed and an enhancement of P2P communication to the framework needs to be done for better support real-time robotic collaboration. To do this, it firstly will examine three popular platforms: JINI, UPnP and Salutation and finalises the architecture of the framework without the JXTA platform. Then it will define the design requirements of this communication enhancement. Finally it will describe the framework’s design and implementation in detail.

5.2 Analysis and Conclusion

5.2.1 Analysis

The JXTA project aims at establishing a set of P2P communication standards so that P2P applications can follow these standards and peers from these applications can communicate with each other. The JXTA communication protocols are defined by using XML technology, which is like what Jabber has done [jab] [Jab01] and enables messages being delivered crossing different communities such as Gnutelle[gnu], SITI[set], and so on.

The JXTA platform has over 200 classes and its size is 5.45MB (downloaded on 15/11/2002, this version was built on 23/09/2002) and the Shell size is 754KB, which is too big for robots that have
limited resources. The JXTA platform implements all six JXTA communication protocols, which is too heavy and unnecessary for robot collaboration in the following aspects:

- Peer Membership Protocol enables robots to form a self-organized group and enables robots from other groups to join in this group. When robots join together to carry on an application, it means that they have to agree to the common protocols to share the resources. In this project, every robot installs the framework and the framework has defined common protocols to enable the robot to share its resources. Therefore, it is unnecessary to define a protocol for group membership. The robots are classified as small devices for being constrained and it would not be a good practice to let the robot to perform two different tasks at the same time.

- Time is the most critical issue for real-time cooperation. The framework is designed to support the real-time cooperation. In this environment, the number of robots in the system is not so big. These robots are configured to the same network before they are assigned to the application. Hence, they are able to communicate with each other directly and do not need to search resources on other network. Peer Endpoint Protocol is for the peers to find the routes across different network and is also not needed for the framework.

The JXTA platform uses pipes as the communication channels to send and receive messages between services and applications [LY96], which is unnecessary for communications between robots because TCP sockets can meet the communication requirements in the project.

Another key technology of the JXTA project is the Advertisement, which is an XML message. Peers cache, publish, and exchange advertisements to discover and find available resources [virtual]. Based on cached, advertised data and advertisement lifetime, the JXTA Resolver enables peers to quickly find a correspondent peer who is the closest. This mechanism leads to the redundancy and inaccuracy because each peer has to keep the advertised data in its memory and the content of the advertisement is incorrect if a peer drops off accidentally. In this case, an extra step is added to update the contents of the advertisement stored in the memory of each peer. A robot has limited memory and the high performance is the highest priority. Advertising operation affects the robot’s performance if it has to allocate extra memory for saving advertised data. Because the member size of a robot group is not big comparing to nodes in the network, it does not require advertisements.

Finally, the JXTA platform is only implemented in Java and is not fully implemented in other languages. Therefore, the platform does not allow the communication between services implemented by different programming languages occurs.

5.2.2 Conclusion

The previous section analyses the framework based on the JXTA platform and draws a conclusion that the JXTA platform is not suitable for the robotic application requirements. The initial design was to re-use the JXTA platform so that it can save the development of the low-level communication between
peers. Because the design of the framework aims to allow every component in the framework to be plugged in and out without affecting other components, the replacement of the JXTA platform does not affect the functions of other components in the framework. The rest of this chapter will describe the communication layer, which replaces the JXTA platform, in detail.

### 5.3 Architecture

This section describes the architecture of the Robot Colony shown in Figure 5.1. The architecture is divided into three layers, which are Network Layer - Layer0, P2P Protocol Layer - Layer2 and Application Layer - Layer1, and their functions are independent. Service sits across Layer1 and Layer2 and is responsible for the service searching, which is to discover available resources. Layer 0 sits on the top of the robot’s operating system and contains the Java Virtual Machine (JVM) including Java core class libraries. With JVM, the whole system allows dynamic software deployment and distribution because of Java’s features, which are the Java source code can be compiled by any JVM and the binary code can run on any machine installed JVM. Hence, JVM provides us the tool for software components distribution.

Layer 0 is Network Layer, which deals with the communication between devices from various networks, such as wireless, Internet and infrared. In order to enable a new network to plug into the system, the framework provides the user with a configuration interface, which indicates the location of the new network driver and the necessary communication protocol information of this network for other robots. The interface also allows the user to configure the robot connection method. Then, the framework gets the network information from the configuration interface and puts the information into a broadcast message.

Layer 1 is P2P Protocol Layer, which supports basic P2P functions. These functions should include service registering, service discovery, remote procedure calls, instant messaging, requesting services, and services providing. In order to allow the programmer to debug source code at runtime and to update the system, this layer also supports dynamic object loading.

The Application Layer, which is Layer 2, contains all the robot functions. This layer is flexible and allows the framework to add required services or to delete unnecessary services. It also allows further
Service sits between Application Layer and P2P Protocol Layer. It is like “yellow pages”, which contains the information on services. It interacts with P2P Protocol Layer to reply or to lookup services. It is also used to register deployed services.

The primary goal of this architecture is to keep the framework small, simple and flexible enough to allow robot interoperation in different P2P applications. Therefore, the following design points are addressed in these three layers:

- The Network Layer should handle the communication between different network protocols.
- The Network Layer should allow any new network protocol implementation to plug in.
- Messages from the P2P Protocol Layer should be independent of hardware and Operating System.
- Messages from the P2P Protocol Layer should interoperate over other P2P protocols.
- The Application Layer should allow a new service to be plugged in.
- The Application Layer should response to requests from other P2P applications.
- The relationship between these three layers should be independent, which allows further updating or replacement of any layer.

In order to achieve the above listed goals, it needs to analyse the problems that will have to be resolved. The next section will describe the challenges to the implementation of these three layers.

5.3.1 Challenges

This section will list a number of challenges at the implementation of each layer.

**Challenge 1** The Network Layer supports devices to communicate with each other from different networks. It must resolve the problem of network conflict, for example, a wireless robot interacts with a robot on the Internet.

**Challenge 2** The P2P Protocol Layer supports basic P2P functions and also enables the interoperation with other P2P protocols. The biggest challenge of this layer is to develop a lightweight mechanism, which is able to communicate with peers from other P2P protocols. The communication contents must include commands from Remote Procedure Calls, components transferring and instant messages.

**Challenge 3** The Application Layer supports application services migration. Because there is no standard for the service-programming model, it is hard for the services from different service providers to interoperate with each other.

The rest of the chapter will define the requirements for the framework by addressing the above three challenges, examine the related works, describe the implementation of the architecture and also provide the solutions of the above challenges.
5.4 Requirements

The previous section describes the framework architecture and lists the challenges on each layer. In order to implement the framework, these challenges are entailed into the following requirements described in Section 1.4.

5.4.1 Requirements posed by Heterogeneity

The software framework installed on each robot helps robots to form a group based on varying applications. The group is heterogeneous, which entails requirements on three points:

- The robots in the group are from different robot manufacturers. Their requirements are varied, according to different manufacturing standards and specifications. Therefore, the framework needs to understand different hardware requirements on the robots.

- The robots in the group have different capabilities and physical resource constraints in terms of memory, hard disk, processors, and so on. The size of the framework must be small enough to allow the robot to accommodate new software package. At the same time, the functionality of the framework must be general enough to allow robots to fit into the collaborative environment.

- In this system, the robots interoperate with each other via the network and derive from different manufacturers. The physical network interfaces on each robot follow different standards and have different requirements. Hence, the framework embedded in the robot needs to support flexible communication mechanisms.

In addition to the infrastructure of the group, the software (including each robot’s device drivers and services) is of different types. Each of them has their own interfaces, arguments and requirements. To allow software components to interoperate with each other, the following issues must be addressed:

- How is a service from the robot discovered? The framework must provide a description of each software component to allow other software components to discover and use it;

- How does the robot announce its existence and the services that it can provide to the group members? The framework must allow for retrieval of services from other robots and installation of these services. Having placed the software in the right place, the framework must then be able to invoke the software.

- How do the services from different robots interact with each other? The framework on the robot must provide an interface that allows other robots to control it remotely. Therefore, the service must be of the generic object type.
5.4.2 Requirements posed by Dynamism

The robot is designed to be able to work in different applications, in which there are corresponding groups consisting of different numbers of robots. These robot groups are dynamically formed according to the requirements of the applications. They allow robots freely to join in or drop off (for example, because of a failure). When a robot joins in a group, it does not know how many robots there are in the group and is also not aware what sort of services are available in the group. The robot must be able to adapt itself to the group environment and the application efficiently. In order to make the robot flexible in different application environments, the robot needs to be able to determine the situation of the application. Furthermore, the framework needs to configure itself so that the robot can discover the services available on other robots in the group, and also broadcast what sort of services it can provide.

The application requires that the robots in the group interact with each other. Hence, the framework must determine what arguments all available services require, and make sure that other robots can understand the requirements of the services that every robot provides.

5.4.3 Requirements posed by P2P Network Protocol

The communication protocol between robots in the application group is P2P. In the robot community, robots control each other by sending an instant message (containing a command and its arguments) and exchange services via the virtual network. The framework needs to publish/subscribe the robot services and the information of the robot system. Moreover, the framework must have metadata to facilitate the interchange of data and the interoperability of services.

Robots in the P2P community are equal, which means that they can join and leave or fail the group at any time. Therefore, the framework must be intelligent enough to find out which robot fails or leaves the group and also to figure out which robot is a newcomer and what sort of services it can provide. Moreover, the framework must know what environment and arguments the service needs so that it can provide an interface to utilise the service. Hence, the framework needs to manage its member registry and services list and also needs to update them regularly.

During the exchange of services between robots, the framework needs to be able to re-join in the group if it fails. It also needs to rebind with the service when it is revoked from failure.

5.4.4 Requirements posed by Services Life Cycle

As mentioned before, the robot has limited physical resource in terms of storage. In order to provide enough space for the necessary services in the application, the robot must be able to uninstall services that have not been used. The framework needs to have the recorder of each service utility so that the framework can judge which service will not be used in the application.

In addition, the application needs to update services frequently. The framework can accept new services and can replace an old service with new one. The framework on other robots can recognise this
updating action and can configure itself to interact with it. Therefore, the framework must be able to
describe the features of the upgrading service.

On the other hand, a situation may require an older model, in which robots in the group require
older services. In this case, the robots can find out that the service has been updated and can adjust their
configurations to interact with the updated service. Hence, the framework needs to be able to reconfigure
its setting based on the configuration description provided by the framework on another robot.

5.4.5 Requirements posed by Services Management

The services on a robot can be moved amongst the robots in the group according to the application re-
quirements. In order to manage the services in the system, there are four requirements for the framework:

- The framework needs to be able to dynamically adapt the service and to make necessary config-
  urations on the robot, so that other services on the robots can interoperate with the new service
efficiently. It also needs to determine whether the system meets the pre-request of the service. If
  not, the framework needs to be able to find out where it can download the pre-request package.

- The framework needs to be able to find out whether a robot has the required service. Moreover, it
  needs to determine whether the service is out of date. If the service is out of date, it needs to be
  able to update the service without affecting the robot’s performance.

- The framework needs to be able to decide whether the robot has received the service successfully
  after the robot downloads the service. If the transmission of service fails, the framework needs to
  be able to rebind with the robot so that it can get the service package.

- When the robot requests transferring of a service from another robot, the framework must be able
to ensure that the system has enough space to hold the service.

5.4.6 Design Goals

In order to enable interoperation between robots with different capabilities in a P2P manner and to meet
the requirements described above, the Robot Colony framework needs to:

- Manage the resources embedded in the robot.

- Export these embedded resources to other robots in the group.

- Discover the surrounding available resources.

- Dynamically adapt new resources from other robots in the group.

- Update and delete the resources according to the requirement of application.

- Recognise new coming robots and configure the resources that they can provide.
• Interoperate with each other using the P2P protocol.
• Manage/update the robot lists regularly.
• Rebind the services that revoke from failure.
• Use Efficient Real-time Control, and
• Use Distributed Computing.

In order to provide the correct strategy to implement each layer of the framework, it is necessary to investigate the relative works that have been done. The most popular three platforms are JINI, UPnP and Salutation.

5.5 Related Work

Section 5.2 describes that the JXTA platform acts as the P2P communication and RPC tools between nodes and explained the reason why it is not suitable for P2P communication between small devices. Section 5.3 presents the architecture of the framework. In this section, three popular platforms: JINI, UPnP and Salutation are examined.

5.5.1 Jini

Jini [jin] technology is a system architecture, which does not require any particular operating system or network transport as it is based on the Java technology. Consequently it brings individual, self-contained devices together into dynamic, transparent, and simply connected networks to offer more services and more resources to the end-user over the network.

In a Jini system, devices can be added or removed and alter some capabilities of the system, but this action cannot change the system’s identity or basic usability. Consequently, the whole system becomes more dynamic and flexible. A Jini system requires:

• A dynamically distributed system. The system provides resources for executing Java language objects, communication facilities between these objects, and the ability to find and exploit services on the network. By using Java Remote Method Invocation (RMI), the system provides communication between objects across device boundaries and enables these objects to work together.

• A common language and implementation that enables communication between distributed objects.

• A lookup service. In the Jini architecture, a service can be a computation, storage, a communication channel or a hardware device. Members of a Jini network share services and the service communication follows a service protocol. A lookup service is used to find and resolve the services.
5.5 Related Work

- Discovery and Join protocols. These two protocols are implemented on each device and are used to add a service to the lookup service. They define the way that a service becomes part of a Jini system.

- Leasing Mechanism. This mechanism negotiates between the user of the service and the provider of the service in order to guarantee the access to services during the available period.

Therefore, a Jini system enables:

- Users to share services and resources over the network;

- Users to easy access to resources anywhere on the network while allowing the network location of the user to change;

- Tasks to be built and maintained easily.

5.5.2 Universal Plug and Play (UPnP)

UPnP [upn] is a framework defined at a much lower level than Jini [tec] and allows the control of networked devices. In order to enable the communication among the networked devices, UPnP uses protocols like IP, DHCP, DNS, HTTP and XML described in Figure 5.2. IP is used in UPnP to address the device. The device dynamically acquires an IP address by a DHCP service. When the device picks up an IP address from IP pool, it ‘ping’s to check whether any other device with that address exists. If there is no reply, then the address is reserved by the device. Otherwise, the process will be repeated with a new address, until a free address is found.

HTTP sits on top of TCP and is responsible for the Internet connection. All other parts are built on top of HTTP or its variants. HTTPU and HTTPMU are the variants and extend HTTP for UDP package. The devices descriptions are encoded into HTTP headers and these HTTP headers are encapsulated into UDP packages, which will be transmitted by IP multicast. Devices advertise themselves by periodically sending their description on a Simple Service Discovery Protocol (SSDP) channel. Other devices listen
to these advertisements on the channel and find out the information about resources on the network. Additionally, devices also use the channel to send discovery messages and to receive the matched response from other devices.

Generic Event Notification Architecture (GENA) [upn] enables the devices to send and receive notification by using HTTP over TCP/IP and UDP broadcast. Its formats are in SSDP and also are able to catch the changes in the service status for any event. There is a control point between services, which is used to receive event notification. The notification contains a request from the interested service, a location to send the event to and a subscription of time for the event notification.

Simple Object Access Process (SOAP) defines the use of XML and HTTP to execute Remote Procedure Calls (RPC). It is the standard for RPC based communication over the Internet and facilitates the distribution communication over the Internet. UPnP uses SOAP to control devices and return results or errors back to the control points. In a request SOAP message, there are two parts: the action name to invoke and parameters required by the action. A return SOAP message contains the status, return values and parameters. SOAP allows synchronous RPC calls to change the behavior of devices.

UPnP combines SOAP and GENA. This practice allows devices to interact with each other and enables interesting events to be notified. However, to participate in such a system, the device must support the physical media used by the system, the control protocol used by programmatic interface and user interface, and the common basic protocols set by the vendors. Because all devices are defined their own standards, devices from different standards cannot interoperate with a system, which conforms to another standard. Hence, the standard incompatibility exists in UPnP. Further, these UPnP standards are incompatible with existing Internet standards, consequently, it cannot be used to allow remote access to devices in the network [Sai02].

5.5.3 Salutation

Salutation Architecture is independent of operating system, communication protocol or hardware platform [sal]. It provides information exchange amongst and between devices. It mainly focuses on remote control of heterogeneous devices over the Internet. With Salutation Architecture, devices can form a dynamic network and also leave or join multiple networks at will.

In Salutation, the device always talks directly to a Salutation Manager (SLM) and the Salutation Managers coordinate with each other. Hence, the function of SLM is like a mediator and also acts as an agent to do transactions on behalf of its clients. It not only carries the data, but also defines the data format to be used in the transmission. However, if the devices communicate with each other directly using native protocols, it is unnecessary to use SLMs.

Except for being a mediator, the SLM is also like a registration station. When a device joins a Salutation system, it registers itself and its available services or resources with a SLM. In order to keep the registry available, SLM can be asked to periodically check the availability of services and to report the status. This allows high availability of system services.
In order to improve performance, the framework provides callbacks for the devices to notify of events like data arriving or devices becoming available.

By using SLMs and importing agent concept, Salutation framework has greater coordination than either Jini or UPnP as the SLMs take the work of services registry, discovery and management from the devices. This also increases the interaction performance of small devices.

5.5.4 Conclusion

This section has described three common platforms: Jini, UPnP and Salutation and Table 5.1 describes the comparison of these three platforms. This table shows that RMI technology is used for the interaction between objects on different devices in the Jini system. As what RMI required, there must be a RMI server for the objects registry and for RMI requests and there also must be a web server for the Internet accessing. All clients must follow the Java standards, which is too heavy for small devices because all client devices have to implement the interfaces defined by the Java language and have to install the JVM. Further, it also limits some currently available devices that have not followed the invoking services standards.

The mechanism of the UPnP system is similar to the Jini system. Instead of conforming to the Java standards, it requires all devices to follow SOAP. SOAP is designed for the RPC communication between applications running on different operating systems, with different technology and programming languages. SOAP is widely known and used standard, however, UPnP requires the participated devices to support the physical media and the system still cannot interact with other standard devices.

The Salutation platform allows heterogeneous devices to plug in and to interact with each other. Instead of forcing each device to follow the communication standards, the Salutation platform introduces SLM, which manages each device’s registry, structures the description of each device and acts as a transaction link between devices. The SLM description is actually a structured XML file. SLM is like a service broker in the middle and negotiates the standards conflict between different devices. This technology is very useful for the interaction involving many devices crossing different networks. But, in this project, the number of robots participating in an application is not large, the system is not so complicate and the communication or interaction is direct. Therefore, it neither requires a service broker in the middle to help messages crossing the network nor forces every robot to follow any complex communication protocol. The framework should cope with the RPC communication by using XML metadata description and the implementation of a simple communication protocol for devices comform to different standards.

5.6 System Implementation Programming Language

As mentioned earlier, the framework allows utilising the resources from other robots on the network. Hence, the framework needs to enable new software to be deployed over the network. Once the software
### Table 5.1: Comparison of three Platforms

<table>
<thead>
<tr>
<th></th>
<th>Jini</th>
<th>Salutation</th>
<th>UPnP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advertisement</strong></td>
<td>Jini Lookup services</td>
<td>SLM registry</td>
<td>SSDP and Directory Service Proxies</td>
</tr>
<tr>
<td></td>
<td>and registration.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Update status</strong></td>
<td>Expiry</td>
<td>Frequently update</td>
<td>SSDP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SLM registry.</td>
<td></td>
</tr>
<tr>
<td><strong>Discovery</strong></td>
<td>Querying lookup ser-</td>
<td>Send queries to the</td>
<td>Listen to SSDP multicast channel or con-</td>
</tr>
<tr>
<td></td>
<td>vices.</td>
<td>local SLM.</td>
<td>tact a directory service proxy.</td>
</tr>
<tr>
<td><strong>Access</strong></td>
<td>RMI Proxy</td>
<td>To a local SLM</td>
<td>To a special URL</td>
</tr>
<tr>
<td><strong>Transport</strong></td>
<td>TCP/IP and proxies</td>
<td>Transport independent</td>
<td>TCP/IP and proxies for other transport</td>
</tr>
<tr>
<td></td>
<td>for other transport</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Invoking Services</strong></td>
<td>Follow Java standards</td>
<td>Flexible and leave de-</td>
<td>SOAP. Focus on base level discovery and de-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vice manufacturer to</td>
<td>vice capability query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>define data and proto-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cols</td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Attribute and values</td>
<td>Structured description</td>
<td>XML Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>by SLM</td>
<td></td>
</tr>
</tbody>
</table>
is adapted into the system, the system should be able to use it as its own component. To fully complete this process, the framework should add heterogeneity and dynamism function of the system.

Java addresses the above challenges and supports the following:

- Java Virtual Machine (JVM) deals with the heterogeneity of the underlying hardware and provides a portable execution environment [LY96]. Hence, software can be developed and compiled somewhere else and transferred to the robot to deploy new applications and to extend and adapt the functions of the framework.

- Correspondingly, JVM provides the facilities of data types to allow code and data to be transferred over the network [LY96].

- JVM also provides a reflection API, which can be used to dynamically discover and invoke the interface of newly deployed software.

Hence, the framework in the project is written in Java programming language and the software can be moved and loaded dynamically by implementing the JVM’s reflection classes.

5.7 Communication Layer

5.7.1 Design of Communication Layer

The Communication Layer in the framework is Layer 0 and is responsible for the communication between peers or for the support of different network protocols. This layer contains two components: Network Layer and JVM. JVM is the basic requirement for running the framework and supports software deployment and dynamism amongst heterogeneous robots.

The Network Layer is the middle layer between the hardware and the software. Its responsibility is to synchronise the communication between devices in accordance with different standards. These devices might be Bluetooth devices, wireless devices, Internet devices or other communication devices. In order to build the communication bridge between them, the Network Layer must understand the different communication standards or protocols and be able to interpret either signals or data from these devices to digital data for the Internet, and vice versa. The Network Layer must also be able to allow a new device to plug in and be able to exchange data with it.

5.7.2 Implementation of Communication Layer

This section describes the implementation of the Communication Layer in detail. A Java Virtual Machine is stored in ROM on the robot.

In order to allow the communication between different network protocols, the Network Layer is divided into two sub-levels: Network Level and Physical Level, as shown in Figure 5.3. The Physical Level acts as a physical network interface and connects with communication devices. The Network Level
is on top of the Physical Level. TCP/IP is the most popular network protocol like a universal connectivity protocol and is widely used. When the network is congested, each TCP connection independently slows down and seeks to find the optimal rate while not losing too many packets. It not only optimizes the bandwidth usage, but also makes the whole system operate efficiently. The Network Level is defined as TCP/IP. In order to allow robots to be controlled or accessed via the Internet, a thin layer HTTP is added above the TCP/IP. The Network Level implements P2P network protocol, in which a peer can be either a server or a client. In this level, Java TCP/IP ServerSocket and ClientSocket are used for sending and receiving data. Another reason to use HTTP protocol is that it can easily support other P2P protocols since it is a request/response protocol.

In the Physical Level, there are a number of physical devices, which convert either signals or data into the network data for TCP/IP, according to the robots requirements in the network. For example, if there is a wireless robot in the network, the Physical Level will receive radio signals and translate these signals into digital data suitable for the Internet transmission. On the other hand, the Physical Level can also convert digital data into radio signals suitable for wireless devices. To reduce the robot resource exhaustion and to increase the network speed, a workstation is introduced to take this task. Another reason for introducing a workstation is that it is easy to plug/unplug in a device and would not affect the functions of robots system if there is a new network protocol.

In summary, the Network Layer is divided into two levels and physically shifts the Physical Level onto a workstation that takes the responsibility of conversion between digital data for the Internet and other standard/format data or signals. This architecture also allows robots behind the firewall to be accessible from outside through this workstation by configuring this machine as a router, but it is out of the project coverage.

### 5.8 P2P Protocol Layer

P2P Protocol Layer is on top of the Communication Layer and communicates with the P2P Protocol Layer on another robot virtually. As mentioned earlier, the framework allows resources on the devices to be controlled, managed and programmed. Further, it also enables the robot system to deploy new software over the network connection. Once the robot system gets the software, the system must be able
to load the software dynamically and the software must also be able to interact with other software installed on the system for new tasks. When the new software becomes one part of the system services, the framework needs to register it so that it can be shared in the network. Having noted that it is unnecessary to install all software to a local robot, the framework must provides a mechanism to discover services, to request services, to provide services and to invoke services.

5.8 P2P Protocol Layer Design

As what described in the previous section, the P2P Protocol Layer supports instant messages, RPC, software components or services transferring and file exchanging. A remote execution command and its arguments can be embedded inside a message and unpacked on the execution robot. The software component and services can also be put inside a message and migrated into the required robot. This procedure is different from instant messaging since this message contains binary code instead of text strings and the system must migrate/manage software components and services. Therefore, this layer has three functions. The first function is that it virtually connects to the P2P Protocol Layer on other robot. This part is the essential element for a P2P communication, in which all data pass through this communication channel. The second function of this layer is to deploy new software and to enable the system to migrate it. Finally, this layer must be able to manage the services and utilise the available services and resources from other robots.

Based on the layer’s functions, the design of this layer is divided into three parts. The following three sections describe the design of these three parts in details.

5.8.1 Communication Channel

Having noticed that the information transferred between peers are irregular, the JXTA Messages are based on the XML language and can describe the information of exchanged data. But the JXTA Messages is defined inside the JXTA protocols, which are too heavy for the robotic system. The simple way to replace the role of the JXTA Platform in the framework architecture is to keep the messages as XML documents and then to create a simple communication channel, which has two functions: sending and receiving data using two different ports. The replacement not only has full function of the JXTA platform, but also should not affect other components in the architecture. In order to do that, it is necessary to analyse the function of the JXTA platform in the framework.

The JXTA platform in the framework acts as a communication tool between peers. It receives messages from other peers and separates the messages into the four following catalogues:

- An XML message embedded with objects. This message is not included inside the JXTA Platform and is implemented by the framework.
- An XML message containing robot commands. This message is not covered by the JXTA Platform and is customised by the framework. The framework uses the JXTA instant message function and
embeds the execution commands and required arguments inside the JXTA instant messages.

- An XML message with measurement data including a single measurement value, or a number of values, or a file like an image file. This message is not the JXTA original message. The framework uses the same mechanism of embedding commands into the JXTA instant message to put the measurement value/values into the JXTA instant message. In order to transfer a file, the framework puts the file into a JXTA message package by using the JXTA file sharing function. To distinguish these two contents, a tag in the JXTA XML message is employed when the framework composes a message.

- An XML message holding binary code. The JXTA Platform originally does not support binary code transferring. For the project, it is important and the framework wraps the binary code inside an XML file so that this XML file can be transmitted using the JXTA file sharing.

The above four type messages are constructed for this project purpose. The construction functions are not originally from the JXTA Platform and follow to the JXTA Protocols so that the messages can be transferred between the JXTA pipes. Therefore, the above messages need to be replaced and the construction functions need to be modified so that the messages are pure XML documents understandable for the project. Hence, the functional requirements of the new replacement are to be able to:

- Send and receive messages;
- Wrap the data into an XML message;
- Parse the XML message and distinguish the data types.

The performance requirements of the new replacement are:

- Small size;
- Reliable;
- Requiring less memory.

### 5.8.1.2 Service Engine

As addressed in the previous section, the JVM provides an opportunity to load software dynamically. However, the Java suffers the following shortcomings:

- Java code is not executed as fast as native code.

- Java’s serialization protocol does not support moving one individual object from one system to another system if this object is one part of a software component. In this case, the whole component has to move with the object.
• In order to load class dynamically, it still needs a customized interface to express the functionality of the object.

• Java Garbage Collector deletes all unused objects. Hence, Java does not support object persistence.

• New objects with the same name space can only be created after the Java Garbage Collector destroys the old object.

The design of Service Engine should address the following issues from Java’s shortcomings:

• Service Engine executes I/O oriented tasks, such as control commands to control the robot and data stream to transfer information between devices.

• Service Engine provides robots with a generic component abstraction, in which base classes are defined to allow components interaction. Components implementing this abstraction can be partially moved between devices. To support this, it requires a Service Engine to support temporary component movement instead of fully migrating the whole software component. Additionally, the Service should allow software to migrate, in response to the change in the system. In order to migrate or replicate software on the robot, Service Engine uses explicit binding provided by the Communication Channel. The Service Engine must be able to judge whether the binding is correct and whether the data downloaded has the same size as the original. If any mistake occurs, the Service Engine would not install the software.

• Service Engine is designed to allow any incoming component to execute in the system. However, it can be switched to be secure and only allows authenticated components to run system commands or to carry out tasks on the system. In order to do this, it requires configuring the system to be secure from the default setting and to plug in the security component defined by the Service Engine component abstraction. For this project’s purpose, the secure component is not implemented.

• Service Engine uses a simple method to invoke a remote object.

5.8.1.3 Service Manager

The major functions of P2P Protocol Layer are to enhance services providing and to support service discovery. To achieve these functions and basic P2P functions as well, a number of Java classes are created as APIs. The functions of these classes are to register a service, to discover a service, to request a service and to provide a service. In order for successful communications to occur, both the service providers and the requesters must agree to a common format for the messages being delivered so that they can be properly interpreted at each end [GGKS02]. All robots participating in the communication group install/run the Robot Colony framework and the framework defines and implements a common interface for robots communication. This layer uses XML messages and the format is defined by the framework’s communication interface. These messages embed with Remote Procedure Calls commands,
measurement values and binary objects. The reason for using XML messaging is that XML messages are OS and language independent and can be easily parsed by using available parsers.

The design of the Service Manager bases on the following three functions:

- **Services Migration** In order to migrate services from different providers, a simple interface with basic services is defined and an engine to manage the services is created. If there is a request to replace an existing service, the engine will be notified and check whether the service is running. If the service is running, the engine will ask the system to stop the thread so that it can replace the existing service. To avoid software version conflicts, the registry also must have an entry to record the updated software version.

New software is installed as an instance of object and also is registered in the Service of the framework. The new software finds one another by querying the registry, which returns the reference of the software matching the description. This reference is then used to invoke methods on the looked-up object, using explicit communication channel, to request a service.

- **New Functions Import** The framework enables robots to be suitable for new applications. It is unnecessary for the robot to install all software and instead it requires the framework to call other functions running on other robots to interact with functions of its own.

Figure 5.4 describes an example of importing a new function. The robot has a digital camera and wants to share it. The camera registers itself as an interface with a description. Service receives the registered description and saves it as “yellow page” index. If an application sends a query to the Service and asks who has a Digital Camera. The Service searches its index matching the description and sends the contact details of the Digital Camera to Application. Then the Application binds Digital Camera for using the Digital Camera.

Each Application has two interfaces. One is for querying and receiving from Service and the other one is for binding with services. In order to keep the descriptions from service, the Service must
have an array. And the shared service must have an interface enabling binding.

- **Remote Control** The major feature of the Robot Colony is to support for providing services (software components) amongst networked robots. The feature is supplied by using Service Provider. Figure 5.5 describes how the Filter and three components of the Service Provider cooperate together to allow the remote control. In the Robot Colony, every software component is saved into a directory, which is named with the service name. The Service Provider on the Robot Colony is like a yellow-page distributed directory service. When a service is installed in the robotic system and can be shared, it registers itself into the Service Provider by using the XML description of each service and its attributes.

The Service Provider has three sub-components, which are Service Lookup, Service Description, and Service Activator. The Service Lookup is responsible for checking the service existence, which is requested by another robot. Filter sends the message from the robot to the Service Lookup through the Service Provider. The Service Lookup will search through its table containing the available services information and answers the request back to the Filter through the Service Provider so that the Filter passes the answer from the Service Lookup back to the requesting robot.

Service Description describes the arguments of each service when it is installed on the robot. The function of the Service Description is to check whether the arguments from the requesting robot meet the Service requirements. If the requesting robot wants to invoke the available service, it must send another message containing the service name and its arguments. When the Filter receives the message, it sends the message to the Service Provider. The Service Provider gets the message and extracts the arguments to check them in its Service Description. If the arguments do not meet the requirement in the Service Description, the Service Provider will send a message back to the Filter so that the Filter can notify the requesting robot with *wrong arguments* message. If the arguments satisfy the requirement of the Service Description, the Service Provider will pass the arguments and service name to the Service Activator.

When the Service Activator receives the arguments and service names, it will invoke the service component and wait for the result from the running program. When the program finishes running, the Service Activator sends the result back to the Filter through the Service provider and the Filter sends the result back to the requesting robot. The functions of the Filter are to abstract the information from the XML message and also to link the Communication Channel to connect other robots.

### 5.8.2 P2P Protocol Layer Implementation

#### 5.8.2.1 Communication Channel Implementation

In order to implement the design of the Communication Channel described in the previous section, the implementation is divided into two steps. The first step is to build a simple P2P communication channel
to send and receive data. This step does not cover transmitting the data content and types in order to guarantee stable P2P channel. The second step is to wrap the data into an XML format message and to parse the message into right catalogue. The following will describe these two steps in details.

A simple P2P Communication Channel

Each node acts as both a server and a client. The communication channel has two virtual connections: one is for the server and the other one for the client. These two virtual connections exist on the node and run separately. They are also two threads, on which two Java TCP sockets are used.

The client socket is different from the server socket, which uses any available port. The server socket requires a specific port that the clients can bind to. The user of the robot must specify the port for the server in the configuration file. When the framework is initialized, the configuration file will be read and the server socket will be created with the configured port.

Challenge 1 and Solution

One of the important requirements of this communication channel is to reduce the usage of memory. The data transmitted between sockets are XML messages and XML messages have multiple types. The socket is used to transfer different XML messages. However, because XML specification does not support multiple root elements, the socket connection has to be closed after having transmitted one XML message and a new socket connection has to be created. This practice does not meet high performance requirement. Hence, the connection socket must be kept opened and must be reused for another transmission.

The easy and popular solution is to put an end to the XML message so that the parser can recognize the end of the XML message.

XML Message Parser

There are four types of messages used in the framework, which means there are four data types. The XML Message Parser must be able to wrap the data into the respected XML message and put the end of each XML message. The parser must also be able to extract each message and parse the message back
to the original data type.

WrappedOutputStream.java and UnwrappedInputStream.java are created for wrapping and unwrapping the data. In the WrappedOutputStream class, there are four functions, which are responsible for wrapping these four data types separately. In order to distinguish them, one tag is employed for describing the data type. In order to put an object class into an XML message, the object is first put into the ObjectOutputStream, then this ObjectOutputStream is serialized into a FileOutputStream, finally the FileOutputStream is embedded into an XML file with the right description tag and the object class name by using an open source project Xerces [xer]. When the XML file is constructed, the file will be sent to the Communication Channel to send out. Listing 5.1 shows the source code how the WrappedOutputStream constructs an XML message with an object class.

Listing 5.1: Embed an object class into a data stream

```
try {
   // Create a file OutputStream
   FileOutputStream fos = new FileOutputStream(objectClassName);
   try {
      // Create object output stream
      ObjectOutputStream oos = new ObjectOutputStream(fos);

      try {
         // write the object
         oos.writeObject(passedObjectClass);
         oos.flush();
      } finally {
         oos.close();
      }
   } finally {
      fos.close();
   }
} catch (Exception e) {
   System.out.println(e);
}
```

In the UnwrappedInputStream class, there are also four functions, which are used to unwrapped the incoming data stream into the respective four types data by recognizing the XML description tab. To decode the data for object class, the parser firstly needs to abstract the class name, and then the parser passes the class name to ObjectInputStream. The ObjectInputStream writes the object into a FileInputStream to story the object into the system. To be able to read the object data into the object, a CustomizedObjectInterface is created so that all objects implement this interface. When the UnwrappedInputStream class reads the object from object data, it casts the object into the CustomizedObjectInterface. Listing 5.2 shows the source code how the UnwrappedInputStream abstracts the object from the object data.
Challenge 2 and Solution

Streams are used for the XML message. Because creating streams takes more computing resources on the robot, it is required to reuse the streams created. When the parser reaches to the end of the XML message, it will put the data into a stream.

At compile time, the length of data is unknown. The stream must allow different length data to be inserted and prevent the XML parser from reading additional bytes of data after the end of the message. Input data is not available immediately. The stream wrappers introduce a protocol that allows arbitrary length data to be sent as separate, localized input streams. While the socket stream remains open, a separate input stream is created to wrap incoming data and makes it appear as if it was a standalone input stream. Both Server and Client are separate threads.

Listing 5.2: Abstract the object class from a data stream

```java
try {
    // Create a file OutputStream
    FileInputStream fis = new FileInputStream(objectClassName);
    try {
        ObjectInputStream ois = new ObjectInputStream(fis);
        try {
            CustomizedObjectInterface obj =
                (CustomizedObjectInterface)ois.readObject();
            /* testing whether get the object */
        } finally {
            ois.close();
        }
    } finally {
        fis.close();
    }
} catch (Exception e) {
    System.out.println(e);
}
```

5.8.2.2 Service Engine Implementation

Software Component

The Component interface, shown in Table 5.2, includes methods to support mobility and life-cycle management. It also includes event handlers that are invoked to notify a software component about the change happening in the component.

The init() method initialises the component state and calls another method to start tasks of the component. After the component is initialised, Service Engine calls getDescription() method and returns meta-data, in XML, to describe the function interfaces, non-function attributes and the event handlers.
5.8 P2P Protocol Layer

/** Component interface */

boolean init(vector arguments);
String getDescription(Component ref);

boolean migrate(Component request, Component destinationm, Method callThis);
boolean replicate(Component request, Component destinationm, Method callThis);
boolean die(Component request);
Vector getBindings();
String getOwner();

/** Event Handler **/

boolean registered(String resource);
boolean unregistered(String resource);
boolean updated(String resource);
boolean lookedUp(String resource);
boolean leaseException(String resource);

Table 5.2: Component Interface Descriptions

The getDescription() method also returns a reference to the functional interface. Service Engine uses
this information to install the new component. To install the new component, Service Engine registers
the reference of the component interface and its XML description with Service.

The XML description returned by getDescription() method is used to dynamically discover the com-
ponent in the system. As the whole framework is designed to be extensible, the new component can be
migrated into the system and interact with other existing components. The component does not know the
attributes of other components before it interacts with them. Instead, the components find each other by
using XML descriptions registered in Service.

Non-functional attributes are used to discover the properties of a component, whereas the meta-
description of the functional interface is used to discover and invoke methods on the component. Appli-
cations find the component with the desired properties and use the description of its functional interface
to select appropriate methods to invoke the service.

Listing 5.3: Functional Interface Description

```xml
<interface>
  index_in_interface_vector </interface>
</interface>

<methodName>method_name </methodName>
<arguments>
  <arguments number=argument_number Type=type_of_argument/>
  <!-- List all arguments required here -->
</arguments>
<description> Description of the operation performed by the method </description>
```
Each component implements the component interface (Table 5.2). The methods and event handlers are well known, but other methods or event handlers are described as part of the meta-data returned by the getDescription() method. Listing 5.3 shows the list of its arguments and implemented method names. The JVM looks up the agent description and uses this reflective information to discover and invoke the method by passing the method name and the type of the arguments to the Java reflection API [jav98].

Any extra event handler, in addition to those in the component interface, implemented by the component is described in Table 5.2.

**Listing 5.4: Description of EventHandler**

<eventName>method_name</eventName>
<methodName>method_name</methodName>
<arguments>
  <arguments number=argument_number Type=type_of_argument/>
  <!-- List all arguments required here -->
</arguments>
<description>Description of the operation performed by the method</description>

The methods migrate() and replicate() in the Component interface are used to move and copy the component to other robots. The Service Engine calls these two methods of the specific component, passing it the destination component, the requesting component and the value used.

Any component can be requested to remove itself from the system by calling kill() method of the Service Engine. The Service Engine calls the die() method of the component and passes it with a reference of the requesting component. This enables the robot to manage the software component and to reuse the system resources.

The getOwner() method returns the name of the owner of the component signed by the private key. Since this framework does not consider the security in the project, this method is not implemented. However, it is necessary to list this method into this interface for the future use purpose.

Finally, the component interface provides a method to return the bindings of the component with other components in the system. The next part explains the bindings between components in details.

**Bindings**

The framework allows two or more components to interact with each other by using bindings. In fact, the bindings (shown in Listing 5.6) themselves are special components with extended functions of Service Engine inheritated from Component. With these extended functions, the bindings support the communication between components and sends message to each other. If the connection fails, an exception is thrown.

**Listing 5.5: Description of Event Handler**

// *** Expanded Component Interface ***
MoreEngine extends Engine
boolean connect(Engine src, Engine dest, Vector Arguments,
   Engine suggestion) throw EngineException;
Object sendMessage(Object message) throw EngineException;
boolean migrateRequest(Component request);
boolean replicateRequest(Component request);
boolean close();

The connect() method in the bindings has four arguments. The first two arguments are the reference
of two components interface. The third argument is for indicating the protocol used and the last argument
is from the Service Engine to move the component. The return value of the connect() method is boolean.
“true” indicates the success of connection established. Otherwise, the connection fails.

The sendMessage() method in the bindings is used to invoke methods on the component. This method
takes messages to the destination via the Connection Channel. Listing 5.6 shows part of code between
two components in the Demo of Appendix A.

Listing 5.6: Component Bindings

/* The function name */
String functionName = "takePicture";
/* Find the function name */
Component myCom = Service.lookup(functionName);
/* Find a binding */
Component binding = Service.lookup("binding");
if ( binding == null )
{
   /* Connect the component to remote component has right function */
   binding.connect(this, myCom, new Vector("take a picture"), null);
   /* Deliver the command to the remote component */
   binding.sendMessage(new Message(camera.takePicture()));
}

The function of the Service Engine is to migrate and replicate the components. In fact, it is a thread
in the JVM. Listing 5.7 shows the interface of Service Engine.

Listing 5.7: Service Engine Interface

boolean init(Component agent, Vector arguments) throw EngineException;
boolean migrate(Component request, Component currentCom,
   Component destination, Method callMethod) throw EngineException;
boolean replicate (Component request, Component currentCom,
   Component destination, Method callMethod) throw EngineException;
boolean kill (Component request, Component currentCom);
int size();
Vector componentList();
The functions of Service Engine Interface are the following:

- **init() method.** This method is called to start. Developers develop and compile a component on the local machine, then invoke the init() method of the Service Engine with arguments. If the method returns true, the loading of component is successful. If the method returns false, an exception is thrown.

- **migrate() method.** This method is called by a component to migrate the component. The requested component must specify a destination component with the method to be invoked at the destination system.

- **replicate() method.** This method is called by a component to replicate the component. The requested component must also specify a destination component with method to be invoked at the destination system.

- **size().** This method returns the size of component so that it can guarantee the successful transmission of the component.

- **kill().** This method is used to kill the component when it is not used any more. The Service Engine calls die() method of the component. If die() method returns true, it means that the component agrees to stop its execution and is ready for the JVM garbage collector.

- **componentList().** This method is used to list the components of the Service Engine.

### 5.8.2.3 Service Manager Implementation

The section presents the descriptions of the Service Manager implementation. The Service Engine is responsible for the interoperability between software components. However, the framework allows dynamically deploying software components and the Service Engine cannot cope with the management of these new components. Further, each component might relate to some events. To interoperate with a component, it requires not only the description of the component, but also the descriptions of events related to the component. Hence, the Service Manager is introduced into the framework and its function is to route the component. In Chapter 3, currently available robot control methods are compared and it concludes that event control is the most efficient control method. The Service Manager also uses event handlers to deal with event synchronicity.

Figure 5.6 and 5.7 shows an example of how the Service Manager works with the Service Engine. Figure 5.6 describes how an extended component registers itself and Figure 5.7 illustrates the calling of an extended component.

An extended component is migrated into the system. In order to trace the events, a collector called Event Collector is introduced. This collector is a thread listening to the Service Manager and prepares for the registration event. For the system to migrate Component A, it firstly sends a request to the Service Engine. The Service Engine notifies the Service Manager with a message that Component A requests
5.8 P2P Protocol Layer

---

**Figure 5.6:** Service Manager

---

**Figure 5.7:** Service Handle
to migrate. The Service Engine forwards the message to the Event Collector and waits for its reply. Once it receives a notice from the Event Collector that Component A has been migrated successfully, the migration of Component A has completed.

The Service Manager also works with the Service and the Application Layer to address the dynamism of the framework. A function query is sent to the Service from the Application Layer and the Service redirects it to Service Manager. The Service Manager searches its registry interface and invokes the relevant events in the Application Layer. When these events complete execution, the Service Manager gets the result and sends it back the requestor.

In this implementation, a status argument is used to manage the event status.

5.8.2.4 Configuration Facilities

The framework supports the P2P communication between devices. As the nature of the P2P protocol, the device must be either a server or client. The server must provide its port and its address to clients so that they can be connected. When a robot joins a robot team, it must let the rest team members notice its existence and the rest robots also need to notify the robot with their on-line situation and available services so that they can initiate communication. The necessary information must include:

- The communication protocol for both communication sides.
- The destination address on each side.
- The destination gate point, for example port number of HTTP.
- The security setting on each end.
- The authentication or authorization details on each side.

All robots have installed the framework, which defines the base communication protocol as TCP/IP and adds HTTP above TCP/IP, and the communication protocol for all robots are TCP/IP. The project does not cover any issue security issue, but for the further development, the framework has an interface to allow security component to plug in. Listing 5.8 shows an example configuration file of one Koala Robot used in the Smart Device Lab at QUT.

**Listing 5.8: Koala Robot’s XML Configuration File**

```xml
<?xml version="1.0" encoding="utf-8" ?>
<!--
/**
 * ****************************************************************************
 * FILE_NAME: koalaconfig.xml
 * **
 * ****************************************************************************
```
Additionally, the framework allows the robots to be controlled in single user mode or multiple users mode. A configuration facility is designed based on experience of robot control in the lab. For example, the robot can be shared by more than one user and is accessible from the Internet. If one user tests his program and the other users do not know, it can cause confuse and misdirection. Therefore, Single User Mode Function is introduced.

Figure 5.8 illustrates the implementation of single user mode. A robot sends a request with its IP address to bind so that it can control the robot. The robot checks whether it is in single user mode. If it is in single user mode and another user is using the connection, it will put the request into the waiting list so that the robot can process the request later on. If it is multiple users mode, a new connection will be built. If it is in single user mode and the connection is not occupied, it will be treated as multiple users mode. Once the connection is established, “singleMode” becomes true. The wait() method is used to control the process. In order to avoid one process taking all the time, we allow the user to define a timeout interval. If there is no definition, a default timeout - of 10s is used.

5.9 Service

5.9.1 Design of Service

The Service in the framework acts as “yellow pages” of software components. A Component is installed with an interface instance by registering its XML description. The Service manages components by registering, deleting, and updating the registry. By using XML, it manages the properties of each component.
The robot is under constraints. In order to improve the performance of the whole robot team and to avoid one robot taking all tasks and the rest robots are unloaded, the framework must control its services (software components) and must:

- limit the connection time between components. Each connection between components runs into a thread. When the component accepts the connection request from another component, this thread starts to calculate the time. And when the connection between two components has been initiated, the count number of accessing is counted down by 1. If it reaches connection time, the thread will stop the process. Then, the connection will be broken and the accessing number will be count up by 1.

- limit the component accessing number. In order to avoid putting a robot into an overburden situation, the framework allows the user to set up how many connections the component can build. Each time when a connection is set up, the framework counts down the access number. When the access number reaches to zero, the framework refuses any connection request. However, if any connection fails, the access number counts up. Then the framework is ready for accepting connection requests.

- allow another component to notice its status. To be able to control a robot, it is necessary to master the status of the robot and its installed components and to track the process. The framework sets up a temporary file when it forks a thread to process the task of a component. When the thread
finishes the processing, the temporary file will be removed. If the framework receives a request for checking the process status, the framework will check the temporary file. If this temporary file exists, the framework treats the process as being on. However, if the process suddenly stops by errors, the temporary might exist. To avoid misleading the framework, and giving wrong feedback, the framework checks the process with setting time. When the framework receives a request for checking the process status and the temporary file exists, it will check the process twice in a short time interval and compare them. If it is different, this means that the thread is still running without error. If there is no difference, the framework will delete the temporary file, update the processing time and reply that the robot is ready for a new process.

- limit the time of each component processing. In order not to “overheat” any robot in the team, the framework allows the user to define how many processes the robot can handle simultaneously. The framework starts a thread whenever it accepts a request from other component, then the thread counts down the number. This setting is optional. If the user does not set the value, the framework will assign its value as the connection time. If the user gives the value shorter than the connection time, the framework would accept the value. If the user sets the value larger than the connection time, the framework will resign the value as the connection time. However, there is a case that the controller wants the robot to process a task longer than the connection. To enable this case occur, an extra attribute is added. When this attribute is set, the framework will ignore the setting time comparison and follows the setting.

Listing 5.9 shows the content of a component message, which the robot broadcasts in the network. It contains the server IP address and the port number to accept the request. It also describes the component name and its description. Finally it indicates the robot’s capability of how many robots the robot can provide the service to.

Listing 5.9: Component Properties XML file

```xml
<component>
  <name>turn</name>
  <description>turning</description>
</component>
```
5.9.2 Name Space and Time

The Service manages the component instances. Since it allows many instances of one component, the biggest challenge is how to find the right instance. “time” attribute is the best description of component instances [THR97]. Therefore, the time of each component instance for identification is defined when the instance is initiated.

As what specified in the Java Programming Language, one component can have many instances and these instances cannot have same name space. To avoid this happens, different objects can be initiated and inside each object an instance can be created. Hence, there are a number of instances with same name space, but they are different objects.

5.9.3 Service Implementation

To implement the Service design, there are five functions described in Listing 5.10 needing to be addressed:

- **register().** This method is used to register an instance of a component. To efficiently manage the resource, the time when the component expires is specified.

- **unregister().** This method is opposite against register() method. However, it does not require the expiration time.

- **update().** This method is used to update the component information. To find the right instance, it requires the expiration time.

- **expired().** This method is used to delete a component instance from the registry. If an instance has existed for a while and has not been active, the system will remove that instance.

- **lookup().** This method is used to search the Service index and return the relative information. It has three arguments, but only needs two arguments: one is the resource name in string format and the other one is the time to locate the right resource. The third optional argument is used to
improve the efficiency of the discovery. If the software has installed on local machine, the Boolean value should be true because the software must be registered. When the service receives a query, it must check whether the software has been registered locally. If not, it will redirect to other robots.

### Listing 5.10: Service Interface

```java
boolean register(String resource, Time expireTime);
boolean unregister(String resource);
boolean update(String resource, Time expireTime);
boolean expired(String resource);
String lookup(String resource, Time expireTime, boolean registered);
```

The implementation of this part is similar to the JXTA message broadcasting based on IP addresses. The message is in XML and the registry is refreshed in 30 seconds intervals by default.

### 5.10 Extension Services

Layer 1 is Extension Services. The framework allows services to be migrated and replicated. A new function is installed with the device by registering a reference with the Service, which extends the registering event handlers in the Service Manager. This Layer is designed for more applications development and is an interface between the application development and the framework.

### 5.11 Discovery and Caching

The lookup query runs through every device on the whole network. The system caches the entries if one matches the desired description. The description of entries is saved in the local registry to avoid future searching.

### 5.12 Conclusion

This chapter describes the design and implementation of the communication channel. It analyses the role and function of the JXTA platform in the framework and indicates that the JXTA platform is not suitable for the P2P communication between small devices, such as robots, because it consumes large system resources. To replace the role of the JXTA platform in the framework, this chapter lists the functions required and also compares three successful communication applications, which are JINI, UPnP and Salution, to judge the communication architecture of the framework. Finally it presents the framework communication layer architecture and its implementation.
Chapter 6

Evaluation

The project aims to design a software framework for a robotic system with the P2P collaboration. This chapter will describe the evaluation approaches for the framework and its extensible features. The implementation of the framework is divided into two stages: the framework based on the JXTA platform and the framework with its own customized P2P network interface.

The evaluation will address these two stages, which are the interaction performance of the framework based on the JXTA platform and the communication channel developed in the second stage. Therefore, the evaluation of the framework is divided into two parts and conducted separately based on the implementation stages.

The first part focuses on the functional evaluation of cooperation between robots or a robot operator and robots. This part will test Advertising, Data Transfer, Dynamical Class Loading, and Remote Execution. The second part examines the performance and functionality of the customized P2P communication channels on the framework. This part will compare the customized communication channel to the JXTA P2P communication protocols to demonstrate whether the customized communication channel is suitable for the cooperation between robots in the robotic system.

6.1 General Evaluation of the framework based on the JXTA platform

The framework is built on the JXTA platform, which provides the basic robust tool for P2P communication between robots. A framework needs to run on the right platform, use the right programming language, and support the right standards [FSJ99]. The JXTA platform provides the basic P2P communication functions between devices. Most available P2P applications only provide a specific platform, which is not suitable for the project requirements. The design target of the JXTA platform is to develop a P2P protocol, on which developers can base development of their own applications. In this project, communication between robots is a direct P2P type provided by the platform.

The whole framework is written in the Java programming language, which is an object-oriented language across different platforms. The major communication method between robots is the message
exchanging in XML format according to the JXTA Messages standard or binary data transfer. Therefore, this framework supports heterogeneous interactions between robots.

The standard approach for evaluating software is to make a checklist of features that the software must support [FSJ99]. The following sections will describe the testing environment, the evaluation of the framework functionality and its results.

6.1.1 Testing Environment

In this evaluation, a robot and a PC are employed to form a direct P2P application. Both of them are installed the framework and are on the same network - QUT intranet. The operating system on the Koala robot is embedded Linux 7.3 without X Windows and the kernel version is 2.4.19. In order to run the framework and the JXTA platform, Sun JVM 1.3.1 is installed on the Koala. The communication protocol of the Koala robot is the sercom protocol, based on ASCII commands. The protocol can be redirected over other means of communication instead of using RS232 serial line. In order to control the Koala robot remotely, a wireless radio kit is employed to send or receive radio signals from other devices. Table 6.1 lists the detail of the Koala Specifications [koa].

The Koala is equipped with PC/104, which is a standard for PC-compatible circuit board modules that can be stacked together to create an embedded computer system. It is similar to a standard PC and is equipped with an onboard Ethernet interface.

The PC in the testing environment runs Windows 2000 and also installs Sun JVM 1.3.1. The JXTA platform and framework are installed on it. From the command console, the PC can ping and telnet the Koala.

In order to build the communication channel between the Internet device and the wireless radio device, another PC is employed as the Bridge Server. This PC is equipped with two network cards. One of them connects to a wireless radio kit, which is used to receive radio signals from other wireless radio kits or to send radio signals to the air. The other one connects to the QUT network to receive or send data packages. In order to build a bridge between these two network cards, the PC runs RedHat Linux 7.3 and installs open source software to pass data packages from one network card to the other one.

6.1.2 Functionality Testing

The framework used for the robotic system has four basic functions to support the cooperation between robots and devices. These are: Advertising, Data Transfer, Dynamical Class Loading, Remote Execution. The robotic software programmer compiles the source code, loads the Java class file into the robot system, and then tests his code online as described in Chapter 1 Scenarios. A robot can also send a command line to another robot and asks it to execute the command. After the robot finishes the execution, it sends the result back to the sending robot. The communication between robots is direct P2P protocol, which is supported by the basic JXTA platform. If the programmer updates the Java class file, which has loaded into Java Virtual Machine, the framework checks whether JVM has already loaded the class. If
<table>
<thead>
<tr>
<th>Elements</th>
<th>Technical Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Motorola 68331 @ 22MHz</td>
</tr>
<tr>
<td>RAM</td>
<td>1Mbyte</td>
</tr>
<tr>
<td>ROM</td>
<td>1Mbyte</td>
</tr>
<tr>
<td>Motion</td>
<td>2 DC brushed servo motors with integrated incremental encoders (roughly 19 pulses per mm of robot motion)</td>
</tr>
<tr>
<td>Speed</td>
<td>Max: 0.6 m/s directly, and 0.38 m/s using factory default PID speed controller; Min: 0.005 m/s using factory default speed controller</td>
</tr>
<tr>
<td>Maximum Acceleration</td>
<td>0.075 m/s² using factory default PID speed controller</td>
</tr>
<tr>
<td>Maximum Slope Traversal</td>
<td>43 degrees</td>
</tr>
<tr>
<td>Sensors</td>
<td>16 Infra-red proximity and ambient light sensors</td>
</tr>
<tr>
<td></td>
<td>4 optional triangulation longer-range IR sensors</td>
</tr>
<tr>
<td></td>
<td>Up to 6 optional ultrasonic sonar sensors</td>
</tr>
<tr>
<td></td>
<td>Battery and ambient temperature</td>
</tr>
<tr>
<td></td>
<td>Motor torque and global power consumption</td>
</tr>
<tr>
<td>Power</td>
<td>Rechargeable NiMH Battery with charge level memory. The battery pack can be easily removed and replaced.</td>
</tr>
<tr>
<td>Autonomy</td>
<td>Approx 6 hours (moving continuously without payload)</td>
</tr>
<tr>
<td>(4Ah batter)</td>
<td>Approx 4 hours (moving continuously with maximum payload)</td>
</tr>
<tr>
<td>Extension Bus</td>
<td>Expansion modules can be added to the robot such as KoreBot n board or Kameleon board or PC 104 boards. Khepera turrets (with local processor) are also supported using an adaptor turret. An accessory deck is provided at the front of the robot for mounting any other custom equipment you wish.</td>
</tr>
<tr>
<td>User Available I/O</td>
<td>12 digital inputs [5..12V]</td>
</tr>
<tr>
<td></td>
<td>4 CMOS / TTL digital outputs</td>
</tr>
<tr>
<td></td>
<td>8 power (open collector) digital outputs [12V 250 mA/output]</td>
</tr>
<tr>
<td></td>
<td>6 analog inputs (10 bit A/D converter, 4.096v range)</td>
</tr>
<tr>
<td>Size</td>
<td>Length: 32 cm</td>
</tr>
<tr>
<td></td>
<td>Width: 32 cm</td>
</tr>
<tr>
<td></td>
<td>Height: 20 cm</td>
</tr>
<tr>
<td>Maximum Payload</td>
<td>3kg</td>
</tr>
<tr>
<td>Radio Transmission</td>
<td>2.45GHz</td>
</tr>
<tr>
<td>Frequency</td>
<td>1.2. to 115.2 kbps</td>
</tr>
<tr>
<td>Radio Transmission Speed</td>
<td>Around 100 m</td>
</tr>
</tbody>
</table>

Table 6.1: Koala Specifications
it is running, the system will fork a thread to stop it and force the Java Garbage Collector to delete the old object (because of the name space requirement in the Java language). Currently, no available P2P application has the functionality of **Remote Execution**. In the Remote Execution operation, Robot A asks Robot B to execute the command by sending the command line with parameters required. Robot B sends the result back to Robot A after it completes the execution of that command. This functionality enhances the interaction between robots.

In the testing environment, the PC is supposed to be another robot. For example, the PC sends a command of turning 90 degrees to the Koala. The Koala receives the command and executes the turning command. When it completes the turning action, it sends a message to notify the PC that it has finished the command execution. The PC receives the message from the Koala and displays the message on the user interface. The command used for the testing comes with the Koala manufacturer to show the positions on the Koala and check whether the Koala has turned 90 degree.

In order to test the framework’s ability of dynamically loading classes on the Koala, a testing class is created and implements the command interface. The class is compiled on the PC and is sent the binary code from the PC to the Koala via the JXTA sending pipe. After having confirmed that the Koala has received the binary code successfully, the PC sends the command, which is the class name, to the Koala to invoke the command. To confirm that the Koala executed the command, the testing class has one line to create a file - testing in the Koala’s `/tmp` directory. We check whether that file exists by connecting the Koala to the monitor.

One of the JVM’s features is that its garbage collector automatically collects unused objects. In the Robot Colony, each command actually is a thread. Once the command is invoked by the JVM, the JVM forks a thread and waits until the command completes. To test that the framework is updating the command classes, the time of the testing class thread is extended to 5 minutes. In these 5 minutes, the framework updating operation from the PC is called. The framework running on the Koala calls the system function to kill the testing command thread. In order to check whether the framework running on the Koala updates the testing command class, the traditional method is used, which is to check the file size and the date when it is updated. Then the command that is the updated command class name is sent from the PC. There is a new file created in the Koala `/tmp` directory.

To test the framework function of **Advertising**, we re-start the network service of the Koala by using the Linux command line, which is `service network stop/start`, and define the framework process as the service running on the Koala. The service starts once the Koala system starts up. The service of **Advertising** is a Java thread running every 20 seconds. When the network service is stopped on the Koala, we check the group members on the PC and the Koala disappears from the group member list. But when the Koala network service starts, the Koala appears again in the PC group member list.

Finally we conduct a more complex testing, which contains a combination of the framework’s four functions. On the Koala robot, we stop the framework and remove the turning command class from the command directory. Then we restart the framework running on the Koala robot. On the PC, we make
sure that the turning command has already been installed. Then we invoke the turning command on the Koala robot from the PC. Of course, the Koala does not have the turning command class. But the framework has the ability to automatically broadcast the message to search for the command class file. The framework running on the PC replies to the broadcasting inquire and sends the class file back to the Koala. The framework running on the Koala robot invokes the command class after it has downloaded the binary class file successfully. We also create a modified turning command source code, which allows the robot to turn in opposite direction. For example, if the command asks the robot to turn right 90 degrees, the modified command orders the robot to turn left 90 degrees. We compile the modified source code and then we call the updating turning command on the Koala robot from the PC. This operation has the following steps:

- The framework running on the Koala checks whether the JVM has loaded the turning class. If yes, the framework calls the system function to kill the thread, which is for the turning class. If not, the framework goes to the next step.

- The framework running on the Koala sent a request to the PC so that it can get the modified command class file. Once the framework running on the PC has received the request, it embeds the modified class into a message and sends the message to the requesting pipe of the Koala. After having sent the message to the requesting pipe of the Koala, the PC awaits the confirming reply from the Koala.

- The framework running on the Koala receives the message from the PC via the requesting pipe. It checks whether the downloading operation is successful. If yes, it sends the confirming reply back the PC.

- After the framework running on the Koala has downloaded the modified command class successfully, it invokes the command and we can see that the Koala turns in the opposite direction.

However, there is a chance that the PC does not have the command class and the framework must be able to stop the broadcasting process to save the resources of the robot.

To test this function, we stopped the framework and remove the turning command class from the Koala robot. We also removed the turning command class from the PC. Then we invoked the turning command. Of course, neither the Koala nor the PC had the turning command class this time. But we defined the TTL time as 60 seconds. After 70 seconds, we logged onto the Koala console as the root and used the Linux command to check the processes.

### 6.1.3 Timing

The communication protocol of the framework is a direct P2P type. It is necessary to test the time consumed by transmitting a package of data cross the network. In order to get the data transmission speed, one extra line is added to print out the current time before the command is sent. After the computer
receives the result from robot, an extra line is added to print out the time interval. The testing environment is the normal network and the testing program sends a package of data to another peer and remembers the time as T1. Another peers get the data and send it back to the sending peer. When the sending peer receives the data, it remembers the time as T2. The interval between these two times (T2 - T1) is the round trip time that the data takes to travel through the Internet.

The testing results are very disparate because of the following reasons:

- The time duration for the Koala to execute a command is unpredictable.
- The network traffic affects the data transmission directly.

### 6.1.4 Performance

The framework that runs is based on the current version JXTA platform. However, the JXTA team was still constructing it and some software in the project was not compatible with the later version JXTA platform. The framework could operate on a normal PC and the Koala robot in the lab. We could not test the framework over a group of robots, which contains more than two robots, because of the robots’ limitation in terms of lab resources. However, we did manage to set up a similar environment on PC as on the Koala.

- **Usability.** The whole framework is written in Java programming language. It keeps all functionality of the JXTA platform. Therefore, it requires Java Virtual Machine and the JXTA platform.

- **Completeness.** The project was originally designed based on the JXTA platform and extends its functionality to support the cooperation in a robotic system. It also provides the facility for robot programmers to debug their code on-line. We completed the design and implementation of the framework based on the JXTA platform. The testing described in section 7.1.2 demonstrates that the functionality of the framework has been totally tested in real time.

- **Maintainability.** The framework is an object-oriented application framework and each function has its own class group, which is composed of a number of classes to finish small tasks. Those tasks are independent and small and involve low-cost maintenance.

- **Portability.** The framework can run either under the Linux environment or Windows operating system because the framework is written in the Java programming language and the Java applications can run on any OS that provides the JVM support.

- **Development.** The communication between JXTA platform core and commands is interface communication. If the user wants to develop the framework with more JXTA commands, he can simply write a software component according to the definition of the interface; the framework would dynamically load the command into its system. This feature provides a tool for software developers to develop individual frameworks.
6.1 General Evaluation of the framework based on the JXTA platform

6.1.5 Extensible Features

6.1.5.1 Robot Description Caching

The framework enables robots or devices in the network to share resources. All of the local resources are registered into a description called a profile and robots in the group exchange its profile between each other to get information of others in the group.

Originally the profile was designed to be saved into the file system of the robot, re-writing the updated data over the older data. This approach does not consume too much resources of the robot. However, the framework is designed for supporting cooperation between peers and the profile is frequently used to update, overwrite and search. Hence, saving the profile into the file system leads to inefficiency and consequently limits interoperability of the system, which is not suitable for real-time robot collaboration. Alternatively, the profile can be stored into the memory of the robot.

One of the critical requirements on the robot is to reduction of resource usage on the robot. To reflect this requirement on the framework, the size of the profile is limited as much as possible without affecting the resource searching operation. The most common approach for searching services in P2P applications is to save the information of a robot into the local profile when it joins the group. When the robot leaves the group, it broadcasts a message to notify every peer in the group that it is leaving. This approach speeds up the search operation if there is no limitation on the resource usage. However, it would not be suitable for the robotic system since it has a high resource usage requirement. Hence, the profile mechanism is used to store the local resource information and develop a search interface to search for available services. As mentioned before, this approach does not require too much resources. Another reason for using this method is that the network interface of each device in the group is different. Consequently each search protocol of the network interface is also different. An interface is used to describe the search procedures and later the developers can implement the interface whenever there is any new network interface added.

The search operation is a software component implementing the searching interface. This interface must have three arguments, which are resource name, time releasing researching and a boolean value indicating whether the resource exists. Whenever a resource is being searched, the boolean value determines whether the resource is registered with the local instance. If the boolean value is set to true, the searching operation will stop and returns the handler of that service. If the boolean value is set to false, the searching operation will block and search for the resource on the network. Then the searching component will connect with peers on the network, whereby each peer looks up its own local profile. If one matches the desired description, that peer will return the handler so that the resource can be accessed.

This approach is designed to support multiple network interfaces and allows the developer to develop the software component for future new network interface. So far, we have only implemented the interface for a TCP/IP network. However, this approach not only supports TCP/IP networks, but also supports other network interfaces.
6.1.5.2 User Interface - GUI Compatibility

User Interface - GUI provides a graphic facility to the end user so that he can control the robots easily. The GUI can also be disabled if the OS of the robot does not have X Windows installed. The GUI is one part of the framework. It should be compatible with the JXTA platform and integrate with all the functions of the platform. The framework uses command lines to control the robots. Consequently, the framework GUI enables the user to input the command line and also enables the user to execute P2P commands from the JXTA platform. Further, the JXTA Shell Console is embedded inside the framework GUI.

The framework not only sends the control commands to other peers, but also sends Java binary classes and text files or other format files (like .jpg) across the network. Data Transfer wraps the control commands, Java binary classes and text files into a message and sends the message package to the GUI. The GUI also displays the results of the command execution after the Data Transfer extracts the result from the JXTA message. In order to monitor the status of other robots in the group, the GUI also displays the current online robots. Figure A.1 illuminates the GUI details.

To evaluate the GUI’s compatibility with the JXTA platform, testing is performed on the PC where the framework and the JXTA platform are installed. The command line is used to invoke the camera on the Koala and asks it to send the picture back to the PC. On the GUI, it displays the picture taken by the Koala. Then we switch off the network service of the Koala and invoke the peer display function on the PC, the GUI does not display the Koala on-line. In order to test whether the GUI has the same functionality as the JXTA Shell Console, we input the JXTA standard Shell Console commands to the GUI and check whether the result is same as what the JXTA Shell Console returns.

6.1.5.3 Advantages

The framework has the following advantages:

- Because the framework is totally written in the Java programming language, and the JXTA protocol can be implemented in different programming languages, it can be further developed for different software communication.

- The framework is totally suitable for the cooperation between PCs in a local network, such as in an office environment.

- The functionality of Remote Execution helps the collaboration in a distributed system.

- The framework is also used as a basic system for a robotic system and brings a new solution to traditional robotic system architecture, planning and control. The traditional robotic research is divided into two stages: offline planning and online control. The framework combines these two stages to avoid any inaccurate planning.
• The framework also provides P2P developers with a basic tool for developing their individual applications.

6.1.6 Summary

In this section, the evaluation of the framework built on the JXTA platform is described. The evaluation mainly addresses the functionality of supporting collaboration between peers in the group. The framework has successfully completed implementation of the four function design of the framework, and has also demonstrated the framework objective: to provide a facility to support P2P collaboration between peers.

6.2 General Evaluation of the communication Channel

Previous sections focused on the evaluation of the framework functions, which support the cooperation between peers in the group. Since the JXTA platform is too heavy for the robot (considering of its resource consumption and the restriction of the robot resources), a simple customized P2P communication channel is developed. This section will describe the evaluation of the communication channel. We conduct the evaluation by using a traditional method, which is the quantitative and functional checking list. The output of the evaluation also indicates the efficiency of the communication channel. Finally we compare the communication channel with the JXTA platform.

6.2.1 Functional Checking List

Chapter 5 describes the implementation of the extension framework. The framework consists of three layers, which are Application Layer, P2P Protocol Layer and Network Layer. Service sits between Application Layer and P2P Protocol Layer. In order to demonstrate and evaluate the function of the extension framework, an application - Camera Navigation followed is presented.

The Camera Navigation application allows the robot to find the target quickly by assigning the task onto a powerful PC instead of processing everything by itself. The tilt camera on the robot takes pictures periodically. The Network Layer binds a powerful PC and sends the images to the PC. The PC processes the images and sends the result back to the robot via the binding channel. In order to complete the procedure, three steps are required.

• The robot invokes the camera on it to take pictures, binds the PC and sends the images to the PC. Then the robot waits for the reply from the PC.

• The PC binds the robot and receives the images, then displays them on the screen. Then it processes the images and sends the results back the robot. To make the program simple, the application only shows the images and does not do any image-processing task.

• The robot receives the commands and turns right.
6.2.2 Quantitative Evaluation of the Communication Channel

This section focuses on the evaluation of the whole framework with the customized communication channel in terms of code size, software component migration and software interaction.

- Code size of the framework implementation. The framework is designed for robotic system and hence the size of framework should be small enough to be embedded into the ROM of the robotic system.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Layer</td>
<td>220.3 KB</td>
</tr>
<tr>
<td>P2P Protocol Layer</td>
<td>45.7 KB</td>
</tr>
<tr>
<td>Application Layer</td>
<td>58.2 KB</td>
</tr>
<tr>
<td>Service</td>
<td>26.4 KB</td>
</tr>
<tr>
<td>Total</td>
<td>350.6 KB</td>
</tr>
</tbody>
</table>

Table 6.2: Code Size of Different Layers of the Framework

The framework is designed to adapt the application dynamically and to provide distributed services. It is also used for gathering a group of robots with limited capability to form a complicated application group. Table 6.2 shows the size of each fixed layer of the framework embedded in the ROM of the device. However, in order to run the framework, the JVM must also be embedded into the system.

- New software components are created as an instance, and register their interfaces and attributes in XML with Service. Components find each other by searching the descriptions in XML and are invoked by referring their interfaces (this is the Java Reflection technology). This operation includes registering, searching and unregistering, and is measured by running over 6 virtual machines, and taking the average time. The result is shown in Table 6.3.

<table>
<thead>
<tr>
<th>Operation</th>
<th>100</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>4.9</td>
<td>6.3</td>
</tr>
<tr>
<td>Search</td>
<td>3.2</td>
<td>5.1</td>
</tr>
<tr>
<td>Unregister</td>
<td>2.3</td>
<td>3.7</td>
</tr>
</tbody>
</table>

Table 6.3: Software Components Migration Result in milliseconds

6.2.3 Comparison to the JXTA Platform

The following list shows the results, comparing the communication channel again the JXTA Platform:

- Code Size. The total code size of the communication channel is 350.6 KB, whilst the code size of the JXTA platform is 5.5 MB. This provides the robot for more space for different applications.
• Complexity. The communication channel simply uses the Java Server and Client Socket to build the P2P communication bridges, instead of using Pipes and protocols implementation. This reduces resource consumption and makes the code readable.

• Functionality. Basically, the functionality of the communication channel is similar to the JXTA platform, as required for robotic system cooperation. The JXTA platform has more available functions for the P2P applications, but these functions are not required by the robotic system.

• Performance. The communication channel enforces software component interaction, software dynamism, and device heterogeneity. While the JXTA platform emphasizes communication between devices across different P2P applications, there is a lack of software collaboration ability between these devices.

### 6.3 Conclusion

This chapter answers two questions:

• Is the framework suitable for the collaboration between robots?

• Is the extension framework more efficient than the framework running on the JXTA platform?

The first question is answered by checking the function list from the testing. A number of scenarios have been conducted to guarantee that the proposed functions of the framework have met the requirement of the collaboration. The second question is answered by evaluating an application and comparing the communication channel against the JXTA platform. Hence, the framework meets the project objectives described in Chapter 1 by addressing the challenges defined in Chapter 5.
Chapter 7

Conclusion and Future Work

7.1 Introduction

This chapter will draw the conclusions of the thesis and identify new direction for Robot Colony into the future.

7.2 Conclusion

Recent developments in embedded systems give robots access to the Internet and make them more flexible and capable of performing more complex applications. However, these robots are still limited in terms of size, CPU power, storage resources and memory. Consequently, these robots have only been manufactured for certain specific applications and cannot be re-used for other applications. This presents us with a need to design a better software framework.

Robot Colony, therefore, provides a tool for remote control and for accessing system resources on networked devices. The P2P network improves network traffic and enforces reliability and availability. This thesis has proposed, presented and implemented the architecture of Robot Colony combined with P2P technology. It investigates the challenges posed by heterogeneity, mobility and dynamism of devices and also describes how the framework addresses these challenges. Finally it presents a simple, event-based control system to enable interoperability of devices.

7.3 Future work

Future work is envisioned in the following areas:

- All code for the framework should be compiled to native code or statically linked native code, to reduce code size and improve performance.

- The XML parse should be modified to optimize the update and delete option in the communication channel. Further, this framework uses XML descriptions to search for components, which causes
inaccuracy, consumes more computing resource, and slows down real-time interaction. Hence, a mechanism for looking up the registered component should be developed to enforce interaction of software components amongst different devices.

- Security is one critical issue in the interoperation between devices across the network. The framework so far supports the interaction between devices on the same network behind the firewall. Network security is not listed in the framework design requirement. However, to support the P2P interoperation between devices crossing different network, security must be addressed in the framework.

- Battery operated robots should help minimize power consumption within the system. We have twice performed such demonstrations in the lab. The first time failed because the battery could not supply enough power for the Koala operation. The battery can only last two hours for the Koala operation.

- The JXTA platform was originally selected as the basic platform for the framework. The platform is suitable for P2P communication between different P2P applications. However, it cannot meet real-time P2P communication needs. Having contacted Roberto Bayardo at IBM Almade Lab, it seems uServ and Vinci are the best candidates for the basic P2P platform of the framework. Further, an OS should be developed for P2P applications between small devices. This OS should be designed to allow P2P communication from the bottom layer.

- In this framework, the XML description technology is used to describe the software components for their interaction. However, it needs further investigation of XML descriptions so that they can adapted for different system requirements. Furthermore, a specified XML description standard for interoperation of small device components should be addressed, and a relative parser should be developed.

### 7.4 Summary

This thesis has proposed and evaluated **Robot Colony**, a software framework, to enable functionality within a P2P system. The framework, based on the JXTA platform provides basic P2P collaboration functions between devices. Customized communication channels enhance interaction of software components, either within one system or between different systems. The project has also opened many prospects for future work, including power consumption, P2P OS and XML standards for small device collaboration.
Appendix A

Simple P2P Network Demo

This section will describe a simple P2P Network Demonstration in Smart Device Lab at QUT. This demonstration aims to demonstrate that the network protocol developed can support remote control the robot and can build a P2P bridge between the user and the robot via the Internet. A Koala 6-wheeled robot from the Swiss K-team(Figure A.1) equipped with a pan/tilt camera is used as the controlled robot. This robot carries with two processors, one is for the operating system, which is embedded linux, the other one is used to control the camera. For this project, Java Virtual Machine(JVM) 1.3.1 is installed on the system to run the framework. The commands input from the user PC are sent to Koala via the network and the Linux PC. Figure A.2 illustrates the structure of our P2P network system.

Figure A.1: The Koala Robot with optional pan/tilt camera, Linux PC and wireless kit

In order to access the Koala from the Internet, a PC is employed for the conversion between digital data and radio signals. On the PC, there are two network interfact cards installed: one is used to connect to QUT FIT local intranet like every working station in the lab and the other one links to a wireless kit, which is same as the one equipped on the Koala. The network interface card with a wireless kit is used to receive radio signals from the wireless kit and convert to digital data or is used to change digital data to radio signals for the wireless kit. However, in order to send and receive data from the Internet to the Koala via the pair of wireless kits, there must be a software bridge to link these two network interface cards. The bridge is available on the open source projects and runs on Linux system. Therefore, the PC
has RedHat 7.3 installed.

There are command drivers to control wheel motors, camera and sensors on the Koala from the manufacturer. These command binary are saved in the `bin` directory like the default linux commands. In order to allow the user to access the Koala via the Internet, a port on the Koala is assigned to receive and send data and a Java Class executes the command and sends the result to the standard output for network connection. To be able to control the Koala remotely, there are also two Java Classes required running on the Koala: TCPServer and KoalaRobot. The TCPServer is listening to the assigned port and is responsible for sending data and receiving request(command lines in the project). When it receives a request, it forwards the request to the KoalaRobot object, which is responsible for executing the command related to the request and sending the result back to the TCPServer. Then the TCPServer passes the result back the user.

On the user’s PC, the user configures out the IP address and port number of the robot and an instance of NetworkKoalaRobot is initiated when the framework starts. The instance of NetworkKoalaRobot connects to the TCPServer Object running on the Koala. The TCPServer creates an object of KoalaRobot for receiving the connection. The command from the user are passed to the KoalaRobot object, which controls the robot based on available software objects.

![Figure A.2: Overview of the Demo Structure](image)

**A.0.1 User Control Panel**

In order to show the result of the demonstration, a user graphic interface shown in Figure A.3 is created. On this GUI, the user can control the speed of two motors installed on the Koala base and each motor provides power for three wheels on each side. It allows the user to control two sides wheels either at the same time mutually or at different times, which means two motors can be controlled to have different speeds. Since the Koala power consumption is very critical in this Demo, the GUI also indicates the battery capability. The GUI shows the pictures taken by the camera on the Koala and the image data can be processed on the user’s PC to improve the Koala’s performance. It also displays the data of sensors on the Koala. To enable control the Koal, the user can input the Koala command via the command text field and the result is shown on Command Result Text Area.
In this Demo, we have implemented the Braitenberg theory for automatic control and demonstrated the remote control robots cross the network in the Smart Device Lab, at QUT. The first time was not so successful because the battery on the Koala could not last too long and supply enough power to more functions. The second time was very successful and has shown all features of the framework.
Bibliography


[DR01a] Peter Druschel and Antony Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *HotOS VIII*, Schoss Elmau, Germany, May 2001.

[DR01b] Peter Druschel and Antony Rowstron. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *the 18th ACM symposium on operating systems principles*, Banff, Canada, October 2001.


