Modularity and Mobility of Distributed Control Software for Networked Mobile Robots

Liam Cragg, Huosheng Hu, and Norbert Voelker

University of Essex, Department of Computer Science, Colchester, CO4 3SQ, UK.
{lmcrag, hhu, norbert}@essex.ac.uk

1 Introduction

Recently, more and more intelligent robotic systems have been embedded into the Internet for service, security, and entertainment, including distributed mobile robots. These networked robots have captured the interest of many researchers worldwide [Gol05]. Apart from operation in a hazardous environment (a traditional telerobotic area), networked robots have been applied in a wide range of real-world applications, including tele-manufacturing, tele-training, tele-surgery, traffic control, space exploration, disaster rescue, health care, and as museum guides [HTCV04]).

Networked robots developed using distributed computing paradigms, normally cover large areas, communicating via wired or wireless networks. To effectively control these robots, four control software architectures can be adopted. These are: client/server, in which robots contain knowledge and resources to perform a task and can be controlled from a simple remote client; remote evaluation in which robots have resources, but need to obtain knowledge from a client to perform a task; code on demand in which robots contain resources but act as clients to access knowledge at a server to perform a task; or mobile agents, in which robots contain resources, and a task can be performed by moving a mobile agent containing knowledge and results between robots.

Mobile agents (MA) are a new distributed computing paradigm, which have not yet been extensively employed in multiple robot architectures e.g. [VCMC01] and [Pap99]. As part of wider research into the development of multiple robot architectures for hazardous environments, we have employed mobile agents to implement autonomous and remote control architectures. A number of experiments have been conducted to assess potential functional advantages, specifically through the use of mobile agent mobility. In this chapter we describe our implementations and experimental results in the use of mobile agents for multi-robot architecture development.

The rest of this chapter is structured as follows: Section 2 provides some background information on mobile agents and how they can be employed in the software engineering process in robotics. Section 3 describes two types of mobile agent implementation, i.e. autonomous and telecontrol. Section 4 describes experiments conducted with these architectures. Section 5 is used to discuss the benefits and challenges which we believe MAs offer a multiple robot system developer. Finally, Section 6 provides some conclusions.

2 Mobile Agents

2.1 Mobile Agent Definition

The concept of an *agent* is widely used in computer science, AI and robotics, often with different meanings. In this chapter the term *agent* is used to mean a software component which exists within a bounded execution environment, but which can control its own execution and interaction i.e. it may be autonomous, dynamic and intelligent (able to learn). A *mobile agent* can move (or migrate) between computing nodes within a distributed computing system.

2.2 Distributed Computing Architectures

In addition to the mobile agent architecture, there are three other main types of distributed computing architecture. These are:

- *Client/Server* - as shown in Fig. 1(a), in which a server (S), has knowledge (K), and resources (R), to perform a task. A client (C) sends a message to the server requesting task execution. The server performs the task using its knowledge and resources, and returns a result to the client.
- *Remote Evaluation* - as shown in Fig. 1(b), in which a client has knowledge, while a server has the resources to perform a task. For the client to obtain the result to a task, it must send its knowledge to the server. The server uses this knowledge and its own resources to perform the task, before returning a result to the client.
- *Code on Demand* - as shown in Fig. 1(c), in which a client has resources, while a server contains knowledge to perform a task. For the client to obtain a result to a task, it must send a message to the server to request knowledge. The client uses the server's knowledge, and its own resources, to perform the task and obtain a result.

In the *mobile agent architecture* in Fig. 1(d) an agent server (AS), is host to a mobile agent (MA), which contains knowledge and results from previous tasks, but not the resources it requires to perform a new task. Another agent server contains the required resources. Instead of forwarding knowledge to the new agent server, the mobile agent moves to access the resource. When the

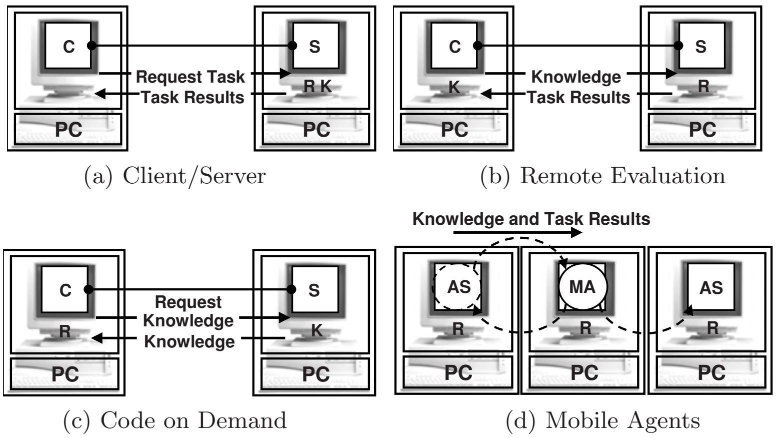


Fig. 1. Distributed Computing Architectures

mobile agent has completed its task, it can remain at the new server, or move to another agent server carrying its knowledge and task results.

2.3 Mobile Agent Environment

The execution environment within which mobile agents operate has the following components and characteristics:

- *Execution Platform* - mobile agents exist within an execution environment, provided by multiple agent servers as shown in Fig. 2. Agent servers provide *places* in which an agent can reside and interface with functionality provided by the server computer.
- *Programming Language* - most agent servers and mobile agents are written in Java which as an interpreted computer language can execute on a variety of heterogeneous computer platforms.
- *Location Identification* - multiple host computers can provide a distributed (but unified) mobile agent execution environment. Each computer can host a single agent server which can be uniquely identified using the host's IP address. An agent server may contain multiple places and agents.
- *Programming Characteristics* - basic mobile agents are lightweight computing components because many functions are available from a function library provided by the agent server, including agent creation, destruction, cloning, migration and fault tolerant storage to persistent media e.g. a hard drive.
- *Execution Characteristics* - most agent servers make extensive use of multi-threading to execute agent activity. Mobile agents run in an independent thread, allowing multiple agents to execute concurrently. A

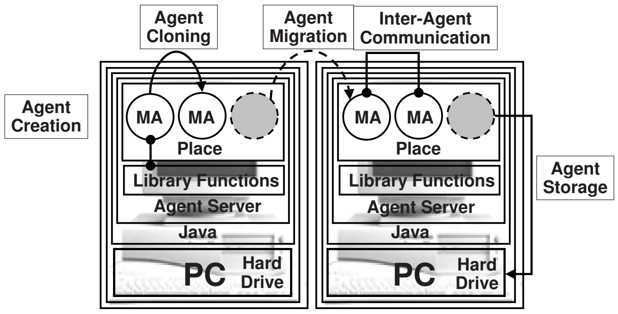


Fig. 2. Mobile Agent Execution Environment

mobile agent itself may be multi-threaded so that it can execute parallel activities.

- *Agent Communication* - Mobile agents can make use of distributed communication mechanisms: sockets (plain or SSL), RMI and CORBA. Mobile agents may also migrate to a resource location to communicate locally and reduce network load.

2.4 Mobile Agent Applications

Mobile agents have been used in a range of applications, as described in [Mil99], [GCKR02], [Lan98] and [MSS01] to play the following roles:

- *Implementation of Code or Actions at Remote Locations* - mobile agents can encapsulate protocols, data, actions, processes and temporary or permanent software and distribute it to remote network locations in order to update system functionality or perform a task.
- *Reduction of Communication and Latency across Unreliable or Limited Bandwidth Communication Mechanisms* - by migrating to a remote location and conducting a sequence of actions locally which would otherwise involve communication across an unreliable communication mechanism.
- *Provision of Personalised Virtual Presence for Users* - MAs can be used as a software embodiment of a user because of their ability to perform intelligent actions remotely and asynchronously.
- *Provision of Dynamic Network Management or Monitoring Functionality* - dynamic, intelligent, autonomous, mobile properties allow MAs to manage or monitor computer networks in which the topology of the network may change.

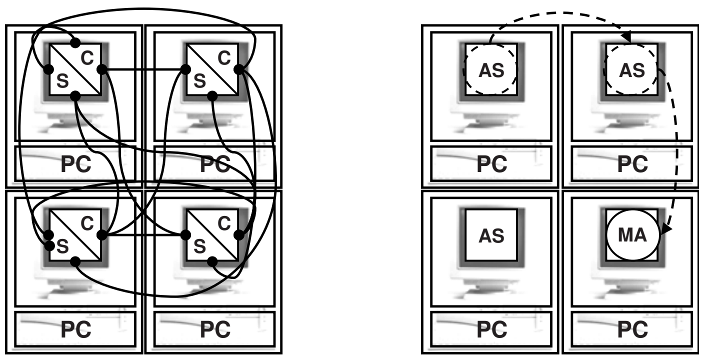
2.5 Mobile Agent Software Development

This section examines the characteristics of mobile agents as a software development environment for distributed robotic systems, in terms of software engineering methodology, security, and testing and debugging.

Software Engineering Methodology - due to their mobility, and ability to move knowledge and task results, mobile agents can provide the functionality found in all other distributed computing architectures combined. This means that in order to provide equivalent functionality to a mobile agent architecture, an architecture would need to be constructed from an amalgamation of all other distributed computing architectures, i.e. client/server, remote evaluation and code on demand.

A simplified view, showing comparable mobile agent, and amalgamated architectures, is shown in Fig. 3. It can be seen from Fig. 3(a), that in the amalgamated architecture functionality at distributed computing nodes is heterogeneous, and dedicated communication links are required to allow task execution. In contrast, in the mobile agent architecture shown in Fig. 3(b), functionality at distributed nodes is homogeneous, and dedicated communication links are not required to allow task execution. Direct communication is not specifically required, mobility can be used instead.

Pfleeger [Pfl01], states that the characteristics of good design in software engineering are component independence, exception identification and handling, fault prevention, and fault tolerance; and that a good way to improve a design is to reduce its complexity. Fig. 3 shows that the mobile agent environment is less complex and more intuitive to the system designer, because there is greater component independence and homogeneity of functionality i.e. mobile agents are modular and easy to understand, functionality is encapsulated and more loosely coupled than in an equivalent amalgamation of the other distributed computing architectures, as dedicated communication links between distributed system components are not required (because mobile agents can migrate to the source of a resource once, rather than needing to communicate across a communication channel multiple times to perform a task).



(a) Amalgamation of all Other Architectures (b) Mobile Agent Environment

Fig. 3. Distributed Computing Architecture Comparison

In addition, because they provide a unified development environment (i.e. provide the functionality of all other distributed computing architectures already built into a single unified environment), this means that a system developer developing a simple architecture in a mobile agent environment, can easily later extend their architecture to perform more complex tasks, without needing to radically change their architectural structure. A mobile agent based multiple robot architecture can therefore be more easily extended than one developed with other distributed computing architectures, or an amalgamation of such architectures.

Security - an issue often raised surrounding the use of mobile agents is security. Reliable interactions between a mobile agent and robot can be enforced and secured by making use of similar types of security measures found in existing e-commerce and Internet applications. These are often made available via the underlying mobile agent platform. Mobile agent platforms can have external and internal security mechanisms.

- External security mechanisms protect mobile agent and agent server interactions from external threats in agent-to-agent or agent-server to agent-server communication. External security mechanisms can include the use of a *Secure Socket Layer* (SSL) employing X.509 certificates and *Message Authentication Codes* (MACS) for authentication and integrity checking of encrypted transmitted data. Internal security is used to protect underlying host computers and legitimate mobile agents from malicious agent attack.
- Internal security can employ many of the security features built in to the Java programming language [Inc05]. Such security features include, codesource-based and identity-based access control policies, in which agents are only allowed access to particular agent, agent server, or host services, as agents operate in secured domains, which shield underlying agent server or host functionality.

Testing and Debugging - a final issue that might be of interest to a DHRI system developer, is how the code of a mobile agent, which interacts locally with a remote robot, might be tested and debugged. Testing and debugging can be achieved using a number of mechanisms. These include Textual and Graphical User Interfaces (TUIs and GUIs), robot simulators, and remote login software.

Mobile agent platforms often provide textual and/or graphical user interfaces for the management of stationery and mobile agents. Such user interfaces allow system developers to monitor the execution of agents at run-time, providing information on the location of executing agents, and their current behaviour.

On a remote robot, which may have no monitor or keyboard, such TUIs and GUIs can still be employed, using remote login and programs such as Exceed [Hum05]. Exceed will allow a user to display on a local PC, TUIs and GUIs executing on a remote robot. This approach can be adopted once an agent has been developed sufficiently for it to be executed on the remote robot.

To increase the speed of testing and debugging at a development stage, as with wider robotics research, testing and debugging can be conducted on local networked PC's using simulated robot software. Modern research robots available with simulation software, while not providing real world test data, will allow initial development, debugging, and testing, to be performed in an accessible and efficient way.

Having described mobile agent technology, the following section describes our implementation of mobile agent based architectures for networked robots; beginning with a description of the motivation for our research.

3 Implementation

Our research has had two main objectives (1) to investigate the development of an architecture to control networked robots in hazardous environments, specifically nuclear decommissioning characterisation [Cra05] and (2) to investigate the characteristics of mobile agents as a development environment for multiple robot architectures.

In pursuit of the first objective, we implemented three control architectures: an autonomous architecture, an implementation of the ALLIANCE architecture [Par98], a telecontrol architecture based on the work of Ali [Ali99] and an AI architecture employing reinforcement learning [SB98]. The aim was to amalgamate these three architectures into a single control architecture that had elements of autonomous and AI control for increased efficiency and telecontrol for safety.

In pursuit of the second objective, we developed a mobile agent based development environment for multiple robots, to enable the implementation of the architectures. We conducted a number of experiments to see if any benefits could be derived from the implementation of these architectures in the MA environment. We specifically focused on the role mobile agent mobility might provide, through experiments in architecture adaptability, fault tolerance and dynamic control allocation. The architectures which employed this functionality were the autonomous and telecontrol architectures, while the AI architecture was implemented to increase efficiency with respect to our first objective. Therefore we only describe the autonomous and telecontrol architectures in more detail in this chapter, for more information on the AI architecture please see [CH05] and [Cra05]

3.1 Mobile Agent Environment for Programming Networked Robots

Our mobile agent environment for programming networked robots is shown in Figure 4. It contains three networked PC's, two running WindowsXP operat-

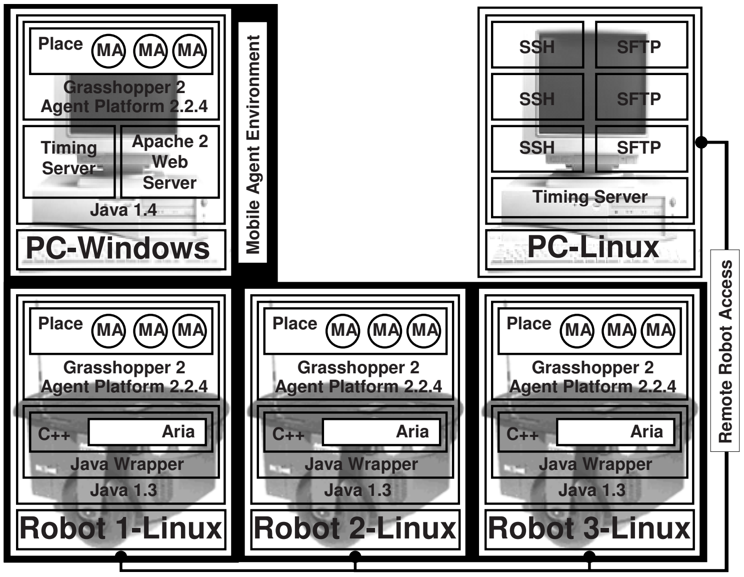


Fig. 4. Mobile Agent Development Environment for Networked Robots

ing system and one running Linux, all contain the Java programming language v1.4. The Windows PCs contained the Grasshopper 2 agent platform which was used to host mobile agents, a timing server to time experiments, and Apache Web Server v2.0 to serve Java mobile agent .class files to Grasshopper agent servers. The Linux PC contained the secure shell program (SSH), to allow remote login, and the secure file transfer program (SFTP), to allow downloading of files, to networked robots. Three ActivMedia [Rob05] Pioneer 2DX mobile robots (each running Linux, and network accessible via wireless LAN) were used to provide a networked multiple robot team.

Grasshopper is an OMG MASIF compliant Java based mobile agent server (which can be purchased from [IKV05]). It provides an execution environment and function library for Java based mobile agents. The function library provides functions for the creation, destruction, cloning, migration, and persistent storage of agents. A mobile agent developer overrides a live() function in which they incorporate their mobile agent's run-time execution code. An Apache Web Server was used to store agent Java .class files. The Grasshopper agent servers obtain .class files from this central repository.

ActivMedia's ARIA robot control software was used to control the low level behaviour of robots. It allows various levels of control to be executed on ActivMedia Pioneer 2DX robots; it is written in C++ but provided with a native language interface via a Java wrapper, which allows commands to be written in Java based mobile agents directly.

The Timing Server is a multi-threaded Java based server, which listens on known ports, and registers the time of any connection. Because it is not possible to precisely synchronise the internal clock of distributed computers, we employed a single timing server during each experiment, to generate consistent times for experiment duration calculation.

3.2 Autonomous Architecture

Our autonomous architecture is shown in Fig. 5. It is a mobile agent implementation of the ALLIANCE architecture [Par98], containing three main types of agent:

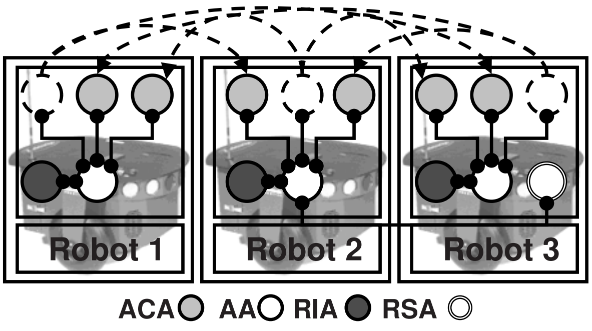


Fig. 5. Mobile Agent Implementation of the ALLIANCE Architecture

- *ALLIANCE Agent (AA)* - the AA contains the ALLIANCE action selection mechanism, and executes behaviour on the robot on which it resides. One resides on each robot in the multiple robot team. AAs engage in: calculation of motivational behaviours (the action selection mechanism), execution of behaviour sets (the behaviour execution mechanism) and management of inter-robot communication (for efficient fault tolerant behaviour) as shown in Fig. 6. Motivation values are calculated using the calculation shown in Table 1 which relies on inter and intra robot input.
- *ALLIANCE Communication Agent (ACA)* - the main purpose of the ACA is to provide an inter-robot communication mechanism between robots. The ACA carries current robot activity and ID information between robots.
- *Robot Interface Agent (RIA)* - a single RIA resides on each robot. RIAs provide an interface to robot functionality for AAs, allowing movement and sensor feedback functions. RIAs shield AAs from underlying robot platforms via a standard control interface.

For more detail on this architecture, please see [CH04a].

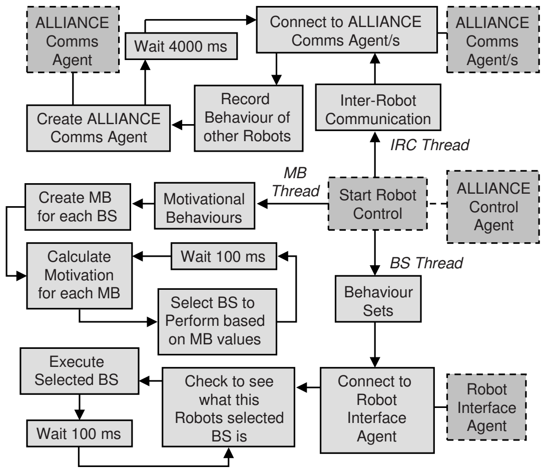

ALLIANCE Comms Agent
Wait 4000 ms
Connect to ALLIANCE Comms Agent/s
ALLIANCE Comms Agent/s
Create ALLIANCE Comms Agent
Record Behaviour of other Robots
Inter-Robot Communication
IRC Thread
MB Thread
Create MB for each BS
Motivational Behaviours
Start Robot Control
ALLIANCE Control Agent
Calculate Motivation for each MB
Wait 100 ms
BS Thread
Select BS to Perform based on MB values
Behaviour Sets
Execute Selected BS
Check to see what this Robots selected BS is
Connect to Robot Interface Agent
Robot Interface Agent
Wait 100 ms

Fig. 6. Flowchart showing ALLIANCE Agent Control Thread Behaviour

Table 1. Code showing ALLIANCE Agent Motivation Calculation

```

//motivation calculation - using ALLIANCE's formal model
private int motivation(){
    int newMotivationValue = 0;
    newMotivationValue = ((motivation+impatience)
            * activitySuppression
            * sensoryFeedback
            * acquiescence
            * impatienceReset);
    return newMotivationValue;}}

```

3.3 Telecontrol Architecture

Our telecontrol architecture is shown in Fig. 7. The architecture contains four types of agent as follows:

- *Local Display Agent (LDA)* - the LDA located on a local PC (PC-L) provides a robot position GUI to show an operator the position of remote robots.
- *Remote Display Agent (RDA)* - the RDA located on a remote PC (PC-R) obtains Cartesian coordinate position information from robots and uses this to update the Robot Position GUI at the PC-L.
- *Local Control Agent (LCA)* - the LCA provides an operator at the PC-L with single or multiple control of robots via a robot control GUI.

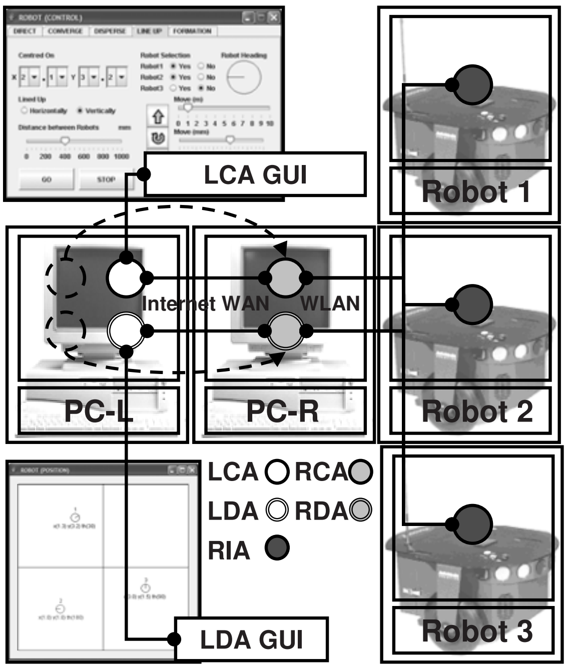


Fig. 7. Mobile Agent Architecture for Multiple Telerobot Control via the Internet

- *Remote Control Agent* - the RCA executes on robots at the remote location, robot control strategies offered by the LCA as shown in Fig. 8.

For more detail on this architecture please see [CH04b].

4 Experiments

Due to their implementation in a mobile agent environment, we were able to supplement the functionality of our autonomous and telecontrol architectures by using mobile agents to provide additional adaptability, fault tolerance, and dynamic control allocation functionality.

To test whether these functional extensions provided benefits over traditional implementations, we conducted a number of experiments, these are described in the following section. We use mobile agent mobility in the experiments in the following ways:

- *Adaptability Experiments* - using mobile agent mobility to automatically update control code at run time in multiple robots.
- *Fault Tolerance Experiments* - using mobile agent mobility to move mission level data from failing system components in the event of their failure.

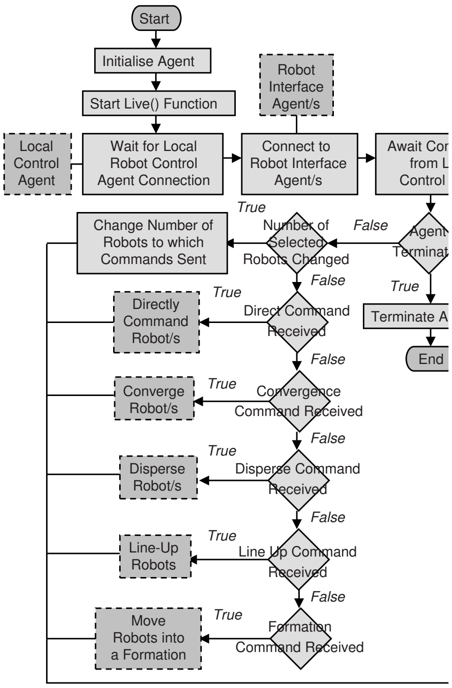


Fig. 8. Flowchart showing High Level Remote Control Agent Behaviour

- *Dynamic Control Allocation Experiments* - using mobile agent mobility to affect the control characteristics of a multiple robot system (i.e. the ability to effectively control distributed multiple robots in the presence of network delay by appropriate positioning of a control agent).

4.1 Adaptability Experiments

In adaptability experiments we employed our autonomous architecture, in the mobile agent development environment with three wirelessly networked pioneer robots. 25 trials were conducted for each set of experiments (a trial for each experiment is described in the following section). In these experiments we used two set-ups.

- *Set-up 1* - used our ALLIANCE mobile agent implementation, in which mobile agents were employed to allow existing ALLIANCE agents, to be dynamically updated by new ALLIANCE agents at run-time.
- *Set-up 2* - used a traditional implementation of ALLIANCE, in which ALLIANCE was implemented in stationary programs, and the secure shell program (SSH) used for remote robot login, and the secure file transfer program (SFTP), used for file transfer to allow updating of control programs on robots.

In each experiment set-up, we conducted a number of activities described in the following section. Table 2 and Figure 9(a), show the architecture adaptability experiment activities for Set-up 1 representing one trial, while Table 3 and Figure 9(b), show the architecture adaptability experiment activities for Set-up 2 which represent one trial.

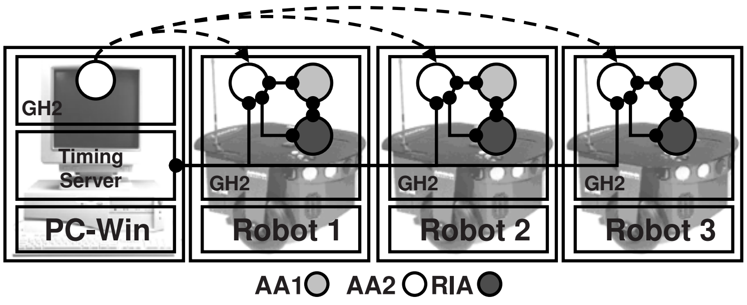
Table 2. Architecture Adaptability Experiment Activities (Mobile Agent Implementation)

Step Number	Activity
1	ALLIANCE agents (AA1) control multiple robots via robot interface agents (RIA)
2	A grasshopper platform is started on an external windows PC, which automatically creates a new ALLIANCE agent (AA2), the time at which this occurs is recorded on the external PC
3	After initialisation, the new ALLIANCE agent automatically makes simultaneous copies of itself on the multiple robots, and removes itself from the external PC
4	The copies of the new ALLIANCE agent, communicate with their existing local ALLIANCE agent on their robot
5	Due to this communication, the existing ALLIANCE agents remove themselves from the robot which they are occupying
6	The copies of the new ALLIANCE agent communicate with their local robot interface agent to take control of their local robot, once a connection has been made, they communicate with the timing server on the external PC, on which the connection time is recorded. The time between Step (1) and Step (6) is recorded as the duration for 1 trial.

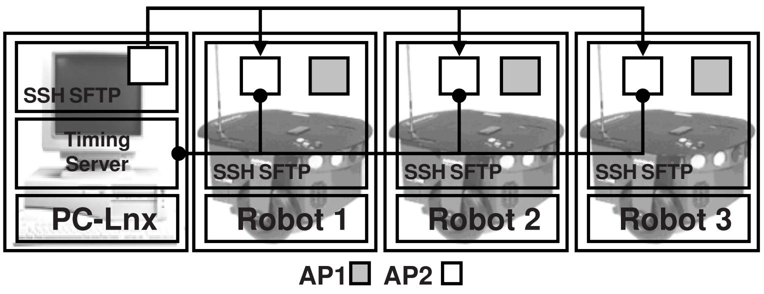
Experiments were conducted using Set-up 1 and Set-up 2, in which teams containing 1, 2, and 3 robots were employed. Architecture adaptability experiment results are shown in Figure 10.

The architecture adaptability results show that using mobile agents, control code can be updated more quickly than when using a traditional approach. Using our mobile agent implementation, a 1, 2, and 3 robot team had its control code updated in, on average 7.19, 8.58, and 9.97 seconds respectively, while using a traditional implementation, the same size robot teams took on average, 25.08, 36.96 and 52.64 seconds to update. The mobile agent implementations are much quicker, because mobile agents can execute in parallel, as they run in independent threads, while in a traditional implementation robots are updated in series.

In our traditional implementation, we tried to use automated scripts wherever possible, to reduce or remove human performance factors, and to speed up and automate the updating process. Despite this, it can be seen that mobile agents in our experiments update code much more quickly than the traditional



(a) Mobile Agent Implementation



(b) Traditional Implementation

Fig. 9. Architecture Adaptability Experiment Activities

Table 3. Architecture Adaptability Experiment Activities (Traditional Implementation)

Step Number	Activity
1	ALLIANCE programs (AP1) control multiple robots directly, the time at which this occurs is recorded on an external PC
2	SFTP is then used to download a new ALLIANCE program (AP2) to one of the robots from the external PC
3	SSH is then used to stop the existing ALLIANCE program, on the robot to which the new ALLIANCE program has been downloaded
4	SSH is then used to start the new ALLIANCE program, on the robot to which the new ALLIANCE program has been downloaded, which then communicates with the timing server on the external PC, on which the connection time is recorded
5	Steps (2)-(4) are repeated until a new ALLIANCE program has been installed and started on all robots in a team. The time between Step (1) and Step (4) (for the final robot) is recorded as the duration for 1 trial.

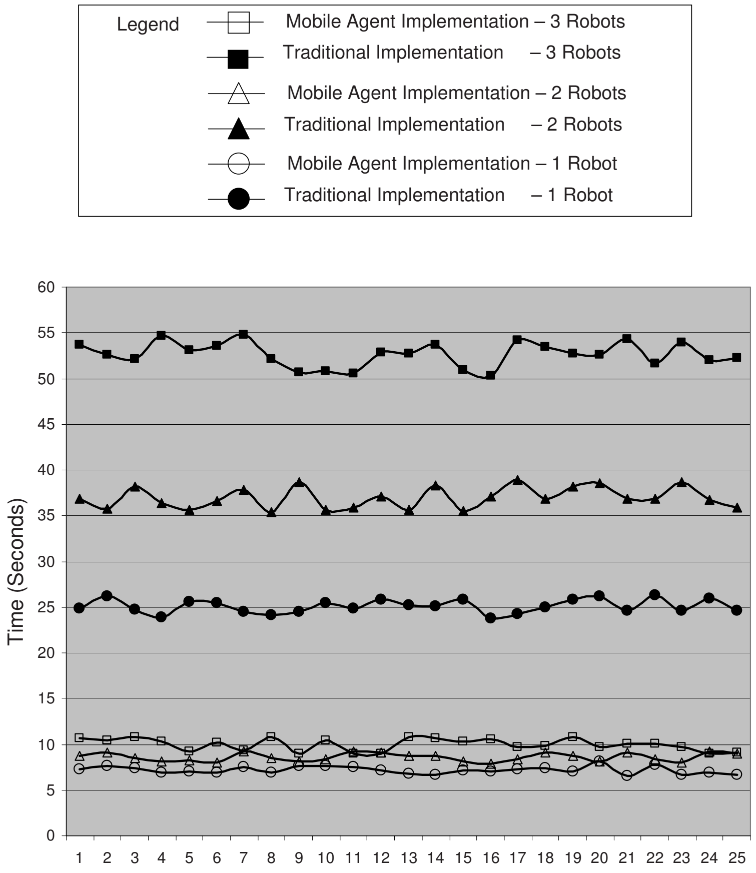


Fig. 10. Architecture Adaptability Experiment Results

implementation. The mobile agent implementation is both quicker, and fully automated, thereby reducing system down time in the event that control code needs to be updated, and reducing the burden on a supervisor.

Due to the parallel execution of mobile agents, one would expect, as robot team number increases, that the mobile agent to traditional implementation control code updating time ratio, would widen, i.e. a mobile agent implementation, relative to a traditional implementation, would be more effective at updating control code as robot number increases.

After architecture adaptability experiments were conducted, we performed mission level fault tolerance experiments.

4.2 Fault Tolerance Experiments

In fault tolerance experiments we employed our autonomous architecture, in the mobile agent development environment with three wirelessly networked pioneer robots. 25 trials were conducted for each set of experiments (a trial for each experiment is described in the following section).

These experiments were designed to investigate how a mobile agent implementation might affect the mission level fault tolerance characteristics of a multiple robot team, (i.e. the ability to retain mission level data in a system despite robot failures). In these experiments we used two set-ups.

- *Set-up 1* - used our ALLIANCE mobile agent implementation, in which an ALLIANCE agent is able to autonomously migrate from a failing robot to a secondary robot in order to maintain mission level data, before migrating on to a new robot.
- *Set-up 2* - used a traditional implementation of ALLIANCE, in which a failing robot loses mission level data when it fails, and is replaced using a new robot with no previous mission experience.

Both sets of experiments were conducted in three robot teams (i.e. one failing, one temporary, and one new robot). In each experiment set-up, we conducted a number of activities described in the following section.

Table 4 and Figure 11(a), show the mission level fault tolerance experiment activities for Set-up 1 representing one trial, while Table 5 and Figure 11(b), show the mission level fault tolerance experiment activities for Set-up 2 which represent one trial.

Mission level fault tolerance experiment results are shown in the chart in Figure 12. The fault tolerance experiment results, show that using mobile agents to retain the state and data of a failing robot takes on average 14.39 seconds, while using a traditional implementation, a new robot can be started in, on average 5.65 seconds. As in the architecture adaptability experiments, scripts were used wherever possible to automate the traditional implementation. This increased the starting speed for the new robot.

These results show that there is some time overhead in using the mobility characteristics of mobile agents to retain mission level data, relative to a traditional implementation. However, the time taken to obtain the mission level data, which is retained in the mobile agent implementation, and lost in the traditional implementation, needs to be taken into consideration.

4.3 Dynamic Control Allocation Experiments

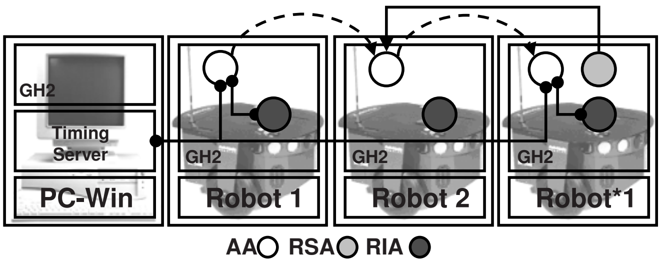
To conduct these experiments we employed our telecontrol architecture, in our mobile agent development environment with three wirelessly networked real pioneer robots, and conducted 25 trials in each set of experiments (a trial for each experiment will be described in the following sections); robots were controlled in a 4x4m lab environment without obstacles, and moved as

Table 4. Mission Level Fault Tolerance Experiment Activities (Mobile Agent Implementation)

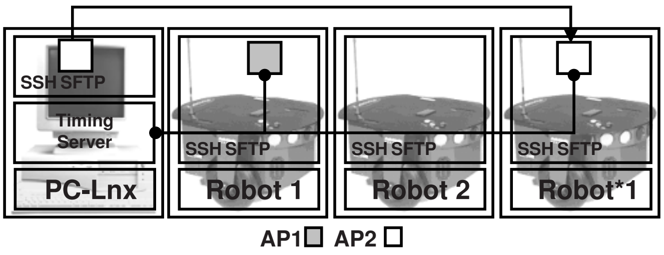
Step Number	Activity
1	An ALLIANCE agent (AA), controlling robot 1 via a robot interface agent (RIA) detects a terminal fault, and communicates with the timing server on an external PC, on which the connection time is recorded
2	This ALLIANCE agent then migrates to robot 2
3	On arrival at robot 2, the ALLIANCE agent waits, and does not attempt to take control of the robot
4	A replacement for robot 1, robot *1 starts. It contains a robot start-up agent (RSA), which communicates with the ALLIANCE agent waiting on robot 2 to inform it that a replacement for robot 1 has started
5	The ALLIANCE agent waiting on robot 2 migrates to the new robot *1
6	The ALLIANCE agent communicates with the local robot interface agent on robot *1 to take control. Once a connection has been made, it communicates with the timing server on the external PC, on which the connection time is recorded. The time between Step (1) and Step (6) is recorded as the duration for 1 trial.

Table 5. Mission Level Fault Tolerance Experiment Activities (Traditional Implementation)

Step Number	Activity
1	An ALLIANCE program (AP), controlling robot 1 directly, detects a terminal fault and communicates with the timing server on an external PC, on which the connection time is recorded
2	A replacement for robot 1, robot *1 starts, and SFTP is used to download a new ALLIANCE program to it
3	SSH is then used to start this ALLIANCE program on robot *1
4	The ALLIANCE program communicates with the new robot *1 directly to take control. Once a connection has been made, it communicates with the timing server on the external PC, on which the connection time is recorded. The time between Step (1) and Step (4) is recorded as the duration for 1 trial.



(a) Mobile Agent Implementation



(b) Traditional Implementation

Fig. 11. Mission Level Fault Tolerance Experiment Activities

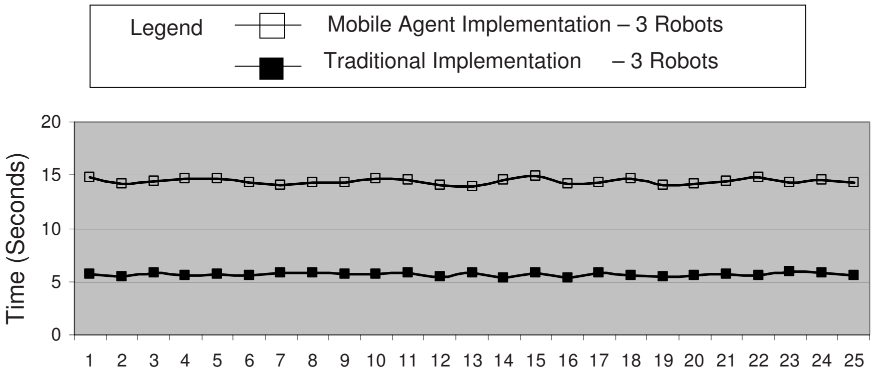


Fig. 12. Mission Level Fault Tolerance Experiment Results

shown in Figure 13 in four multiple telerobot control task types as defined and employed by Ali in [Ali99].

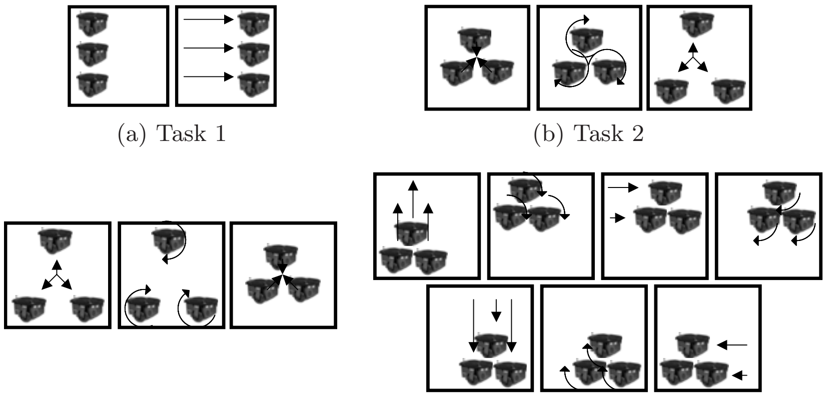


Fig. 13. Dynamic Control Allocation Experiment Tasks

These experiments were designed to investigate, the effect that using mobile agent mobility to dynamically move supervisory group control close to a robot team (at the remote PC), has on control efficiency; relative to the same control implemented close to a supervisor (on the local PC), when Internet communication between local and remote locations is subject to delay. In these experiments we used two set-ups.

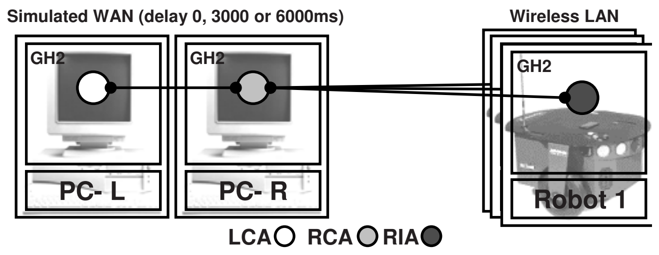
- *Set-up 1* - used our implementation of multiple telerobot control via the Internet, to provide a mobile agent containing supervisory group control for a supervisor to control a team of telerobots; in which the mobile agent is located close to the telerobots at the remote PC.
- *Set-up 2* - used our implementation of multiple telerobot control via the Internet, to provide a mobile agent containing supervisory group control for a supervisor to control a team of telerobots; in which the mobile agent is located close to the supervisor at the local PC.

In each experiment set-up, we conducted a number of activities described in the following section. Table 6 and Figure 14(a), show the dynamic allocation of control experiment activities for Set-up 1 representing one trial, while Table 7 and Figure 14(b), show the dynamic allocation of control experiment activities for Set-up 2 which represent one trial.

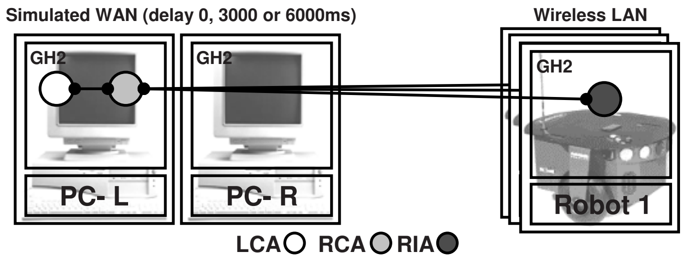
Dynamic control allocation experiment results are shown in Fig. 15 and 16. The dynamic control allocation experiment results show that using mobile agents to encapsulate supervisory group control (and move it close to a robot

Table 6. Dynamic Control Allocation Experiment Activities (Supervisory Group Control Close to Telerobots (Remote PC))

Step Number	Activity
1	Robots are moved to their starting positions (for task 1, 2, 3, or 4, as shown in Fig. 13)
2	The remote control agent is moved close to the telerobots (the remote PC), and simulated Internet delay between local and remote PC, set as either low (no additional delay), medium (low + 3000ms) or high (low + 6000ms)
3	Parameters are set on the control panel GUI so that supervisory group control can be employed
4	A command is sent from the control panel GUI to the remote control agent, which encapsulates the supervisory group control
5	The remote control agent coordinates the low-level control of the robots in the robot group, to move them to the end positions for the specified task
6	Robots reach their end positions.
7	The time between Step (1) and Step (6) is recorded as the duration for 1 trial.



(a) Supervisory Group Control close to Telerobots (Remote PC)



(b) Supervisory Group Control close to the Supervisor (Local PC)

Fig. 14. Dynamic Control Allocation Experiment Activities

Table 7. Dynamic Control Allocation Experiment Activities (Supervisory Group Control close to Supervisor (Local PC))

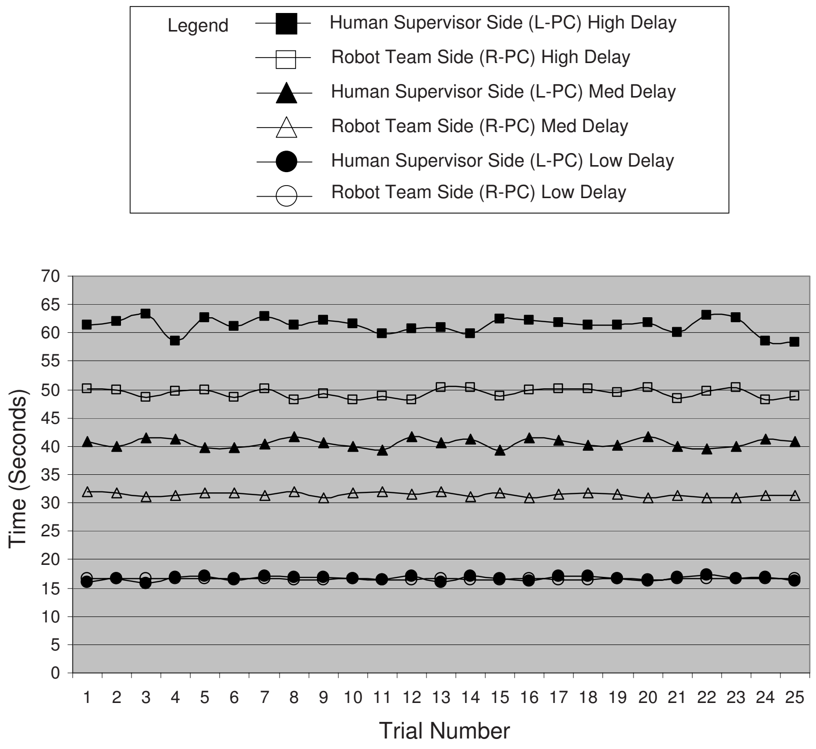
Step Number	Activity
1	Robots are moved to their starting positions (for task 1, 2, 3, or 4, as shown in Fig. 13)
2	The remote control agent is moved close to the supervisor (the local PC) and simulated Internet delay between local and remote PC set as either, low (no additional delay), medium (low + 3000ms) or high (low + 6000ms)
3	Parameters are set on the control panel GUI so that supervisory group control can be employed
4	A command is sent from the control panel GUI to the remote control agent, which encapsulates the supervisory group control
5	The remote control agent coordinates the low-level control of the robots in the robot group, to move them to the end positions for the specified task
6	Robots reach their end positions.
7	The time between Step (1) and Step (6) is recorded as the duration for 1 trial.

team in an architecture in which there is delay in the communication mechanism), allows task execution to be conducted more quickly than if more of the commands are sent across the communication mechanism from the supervisor location.

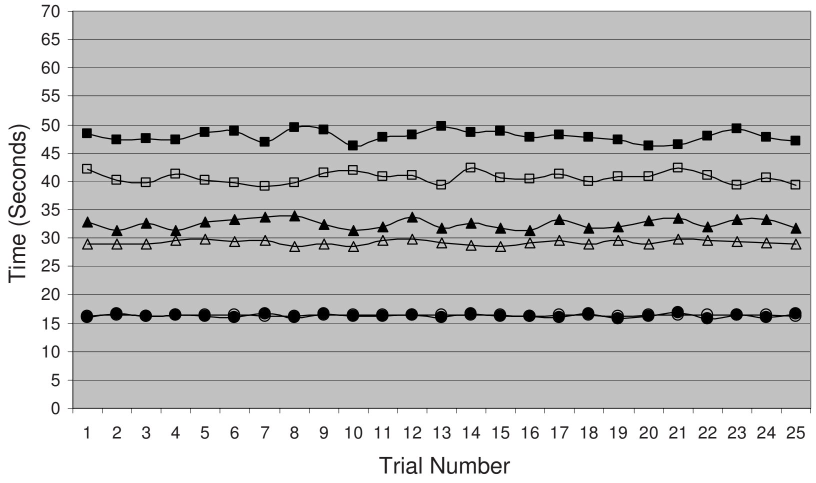
Fig. 15 and 16 show that when there is low delay, there is little difference in time taken, between the supervisor and robot team locations. As simulated Internet delay is increased to 3000ms between supervisor and robot team locations, there is a clear impact on the time taken to complete each of the four tasks. As simulated Internet delay is increased to 6000ms, this impact becomes more pronounced. The difference in performance occurs, because when supervisory group control is located close to the supervisor, more low level control commands must be sent across the simulated Internet WAN, rather than Internet LAN connection (Internet WAN being subject to greater levels of delay).

In addition, because there can be a greater discrepancy between the execution of commands or feedback received across the simulated Internet WAN connection, robot movement is more unpredictable. The main effect of locating supervisory group control close to the supervisor at the local PC is to increase task execution time. This is due to increased command execution delay, and increased command execution/feedback delay variability, which decreases applied control predictability.

Locating supervisory group control at the supervisor location on the local PC, led to a system which was inherently unsafe, and incapable of predictable control. In a real Internet connection, where delay is unpredictable, this would

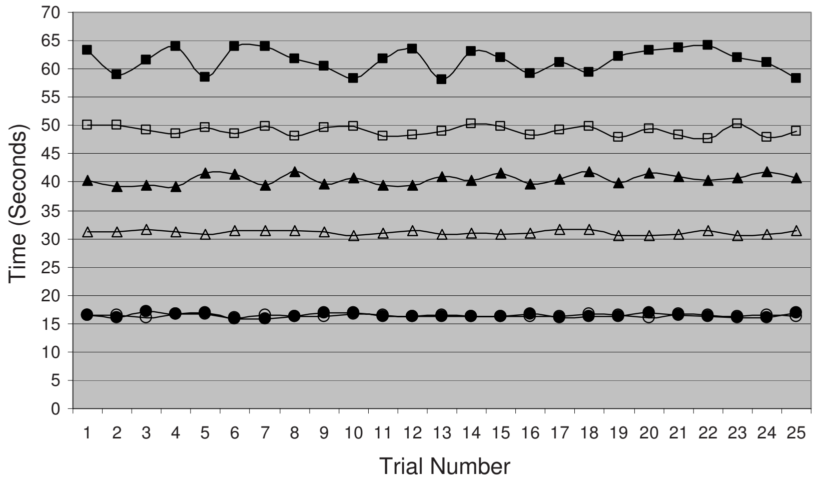


(a) Task 1

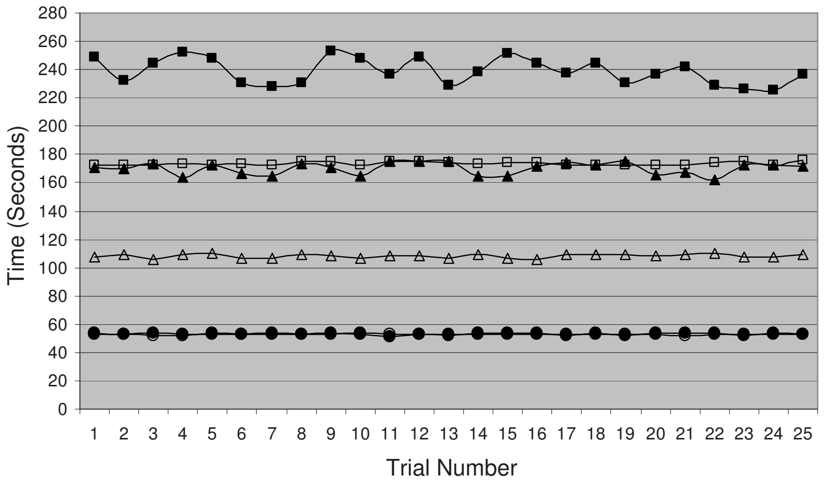


(b) Task 2

Fig. 15. Dynamic Control Allocation Experiment Results (Task 1 and 2)



(a) Task 3



(b) Task 4

Fig. 16. Dynamic Control Allocation Experiment Results (Task 3 and 4)

make control of the robot team more difficult. By moving the control closer to the robot team, using mobile agent mobility, control problems caused by Internet delay are reduced allowing more efficient control of the robot team.

5 Discussion

Based on our experience of mobile agent based multiple robot implementations and experiments, this section discusses some of the benefits and challenges in the use of mobile agents in multi-robot control architectures.

5.1 Mobile Agent Benefits

Functional Benefits

Through experiments using our architectures we have shown that a mobile agent implementation can add the following functional benefits to a multi-robot architecture through mobile agent mobility:

- *Adaptability* - easy automatic updating of control code in a multiple robot system at development or run-time.
- *Fault Tolerance* - ability to automatically and dynamically move important mission data away from failing robot system components for safe storage or reapplication at a later time.
- *Dynamic Control Allocation* - the most effective control can be applied despite network communication delay or changes in network topology, by dynamic MA positioning.

Software Engineering Benefits

MAs provide the following software engineering benefits for distributed robot system developers:

- Because the functionality of all other distributed computing architectures is already built in to the MA environment a system developer can easily extend their architecture to perform more complex tasks without needing to radically change its structure.
- Mobile agents are modular and easy to understand, functionality is encapsulated and more loosely coupled than in an equivalent amalgamation of client/server, remote evaluation or code on demand architectures.

5.2 Mobile Agent Challenges

While mobile agents provide robotisists with many advantages, they have some limitations.

- *Resource Use Challenges* - the mobile agent environment consumes a certain level of resources in the architecture e.g. to maintain agent servers demands that robots should have reasonable processing power.

- *Application Challenges* - mobile agent mobility is useful only when large scale quantities of data need to be moved because there is a time overhead in moving a mobile agent relative to message passing in a networked architecture.
- *Programming Challenges* - system developers must learn the additional structure and functionality of a mobile agent development environment when they may already have experience of client/server, remote evaluation and code on demand applications.

6 Conclusions

In this chapter, we have examined the use of mobile agents for the control of multiple robot architectures, using autonomous and telecontrol architectures developed in a mobile agent based development environment for networked robots. A number of experiments with these architectures were conducted to examine the characteristics of implementing them in a mobile agent environment.

Through this examination and our experience of using mobile agents as a robot architecture development environment, we have found that mobile agents provide a novel software engineering methodology which compares favourably with an amalgamation of other distributed computing architectures. Employing mobile agent's mobility, the adaptability and fault tolerance of a multiple robot architecture can be improved, and control can be dynamically positioned within an architecture at the most appropriate location.

Mobile agents in general make a multiple robot architecture more dynamic and adaptable. They provide the functionality of all other distributed computing architectures combined and as such provide a system developer with this functionality already built in. Combined with a modular software engineering methodology this makes mobile agents an effective framework within which to development multiple robot architectures.

References

[Ali99] K. Ali, *Multiagent telerobotics: Matching systems to tasks*, Ph.D. thesis, Georgia Tech Mobile Robot Lab, USA, 1999.

[CH04a] L. Cragg and H. Hu, *Implementing alliance in networked robots using mobile agents*, Proceedings of 5th IFAC Symposium on Intelligent Autonomous Vehicles, Instituto Superior Tecnico (Lisbon, Portugal), 10th-12th December 2004.

[CH04b] L. Cragg and H. Hu, *Implementing multiple robot architectures using mobile agents*, Proceedings of Control 2004 (Bath, England), 6th-9th September 2004.

[CH05] L. Cragg and H. Hu, *Application of reinforcement learning to mobile robot in reaching recharging station operation*, IPROMS 2005 (On-Line Conference), 5th-14th July 2005.

[Cra05] L. Cragg, *Application of mobile agents to networked robots in hazardous environments*, Ph.D. thesis, Department of Computer Science, University of Essex, UK, September 2005.

[GCKR02] R. S. Gray, G. Cybenko, D. Kotz, and D. Rus, *Mobile agents: Motivations and state of the art systems*, Tech. report, TR2000-365, Department of Computer Science, Dartmouth College, USA, 2002.

[Gol05] K. Goldberg, *Ieee society of robotics and automation technical committee on: Networked robots*, http://www.ieor.berkeley.edu/ goldberg/tc/, 2005.

[HTCV04] H. Hu, P.W. Tsui, L. Cragg, and N. Voelker, *Agent architecture for multi-robot cooperation over the internet*, International Journal of Integrated Computer-aided Engineering, Vol. 11, No. 3, 2004.

[Hum05] Hummingbird, *Exceed homepage*, http://www.hummingbird.com/, 2005.

[IKV05] IKV, *Homepage*, http://www.ikv.de/, IKV++ Technologies AG, 2005.

[Inc05] Sun MicroSystems Inc., *Java homepage*, http://java.sun.com, 2005.

[Lan98] D. B. Lange, *Mobile objects and mobile agents: The future of distributed computing?*, In Proceedings of 12th European Conference on Object-Oriented Programmin (20-24 July 1998).

[Mil99] D. S. Milojicic, *Trend wars: Mobile agent applications*, IEEE Concurrency, Vol. 7, No. 3, Pages: 80-90 (1999).

[MSS01] P. Marques, P. Simões, L. M. Silva, F. Boavida, and J. G. Silva, *Providing applications with mobile agent technology*, Proceedings of the 4th IEEE International Conference on Open Architectures and Network Programming (Anchorage, Alaska), April 2001.

[Pap99] T. Papaioannou, *Mobile agents: Are they useful for establishing a virtual presence in space?*, Agents with adjustable Autonomy Symposia, AAAI Spring Symp, 1999.

[Par98] L. E. Parker, *Alliance: An architecture for fault tolerant multi-robot cooperation*, IEEE Transactions on Robotics and Automation, 14 (2), 1998.

[Pfl01] S. L. Pfleeger, *Software engineering: Theory and practice*, 2nd ed., Prentice Hall, 2001.

[Rob05] ActivMedia Robotics, *Homepage*, http://robots.activmedia.com, 2005.

[SB98] R. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT Press, 1998.

[VCMC01] W. Vieira, L. M. Camarinha-Matos, and O. Castolo, *Fitting autonomy and mobile agents*, Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation, Vol. 2, Antibes-Juan les Pins, France, 15-18 October 2001.