

Application of mobile agents in multiple online robots

Liam Cragg and Huosheng Hu

Department of Computer Science, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, U.K.

Proceedings of the 4th Int. Conference on e-engineering and digital enterprise technology (e-ENGDET), Leeds Metropolitan University, Leeds, U.K. 1-3 September 2004

ABSTRACT

Mobile agents provide the functionality of all other distributed computing paradigms in a unified environment and a natural design philosophy for distributed computing systems. These properties can provide multiple robot system developers with a wide range of options for initial development and future extension of their system and an intuitive and robust design environment. In this paper we present two examples to show how by taking advantage of these properties the adaptability and fault tolerance of a multi-robot architecture (the ALLIANCE architecture) can be extended in multiple online robots.

1 INTRODUCTION

ALLIANCE was developed for fault tolerant control in multiple robot systems. It was originally developed using static code and one-way radio frequency message passing to allow inter-robot communication (1). ALLIANCE has been shown to be effective (2) however real-world applications continue to demand advances in fault-tolerant architectures.

Online robots offer us the opportunity to extend fault tolerant multi-robot architectures like ALLIANCE because through their use of Internet and networking functionality they allow us to develop systems using modern distributed computing paradigms which offer us new design options. One such modern distributed computing paradigm is mobile agents. In this paper we use mobile agents to implement ALLIANCE in multiple online robots in order to show the functionality such an implementation can provide.

The rest of this paper is organised as follows. Section 2 outlines background information on ALLIANCE, online robots and distributed computing paradigms. Section 3 presents a detailed description of mobile agents. In Section 4, details of our implementation environment and architecture are introduced. Section 5 shows experiments conducted using the proposed environment and architecture. Experimental results and analyses are given in Section 6. Finally, conclusions and future work are summarised in Section 7.

2 BACKGROUND

2.1 The ALLIANCE architecture

Fig. 1 shows a simplified diagram of ALLIANCE. Its main components are Motivational Behaviours (MB_i 's) and Behaviour Sets (BS_i 's). MB_i 's receive input from In-Robot Communication (IC) and sensors, and are each linked to an associated BS_i . BS_i 's receive input from sensors and provide output to robot actuators. BS_i 's are independent robot activities which may contain sub-tasks. MB_i 's determine whether their associated BS_i s are selected at runtime based on the robots motivation to perform them. Motivation is calculated using a mathematical formula which takes into account, the behaviour of other robots, the behaviour of the host robot, and sensory information from the environment. Please see (1) for a detailed description.

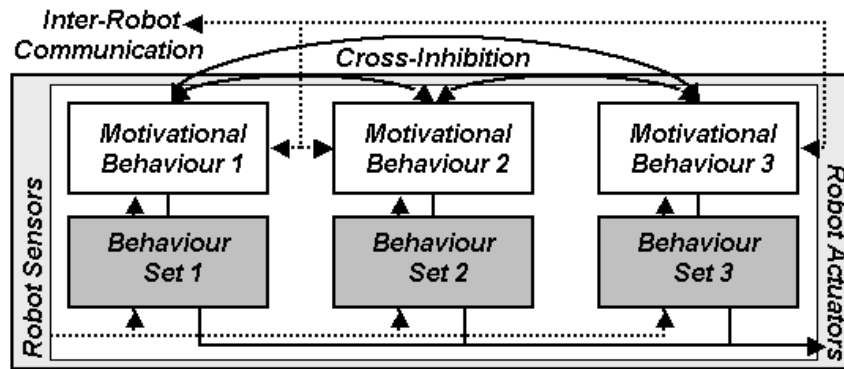


Fig. 1 The ALLIANCE architecture

2.2 Online robots

Online robot systems use Internet software protocols (such as TCP/IP or UDP/IP) for communication and Internet communication infrastructure as transmission hardware. Online robot systems have been developed using single (3) and multiple robots (4). Using widespread, inexpensive Internet technology allows this type of robot system to be widely accessed (e.g. via standard web browsers) and cost effective.

2.3 Distributed computing paradigms

A number of distributed computing paradigms exist with which distributed computing systems such as multiple robot systems can be structured. These include:

2.3.1 Client/server

In this paradigm (Fig. 2a) a server has knowledge and resources to perform a task. A client 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.

2.3.2 Remote computation

In this paradigm (Fig. 2b) a client has knowledge while a server has 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.

2.3.3 Code on demand

In this paradigm (Fig. 2c) 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.

2.3.4 Mobile agents

In this paradigm (Fig. 2d) an agent server is host to a mobile agent 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 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.

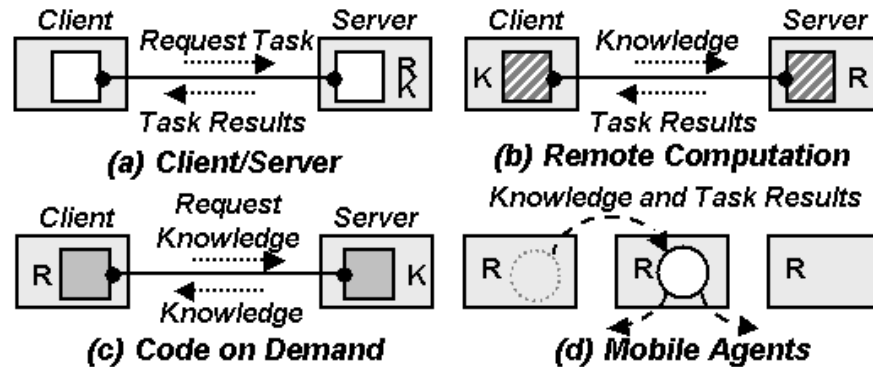


Fig. 2 Distributed computing paradigms

3 MOBILE AGENTS

In the previous section we compared activity in a variety of distributed computing paradigms, in this section we examine mobile agents in more detail.

3.1 Execution environment

Mobile agents exist within an execution environment, provided by multiple agent servers (Fig. 3). Agent servers provide areas known as places in which an agent can reside and interface with functionality provided by the server and host computer. Most agent servers and mobile agents are written in Java which is an interpreted computer language and can therefore execute on a variety of heterogeneous computer platforms.

Multiple host computers providing a distributed (but unified) mobile agent execution environment each host a single agent server which can be uniquely identified using the hosts IP address. Agent servers may however contain multiple places, between which multiple mobile agents can simultaneously migrate. Basic mobile agents are lightweight computing components, because many of the functions which they execute are provided by agent servers in a function library e.g. functions for agent creation, destruction, cloning, migration and fault tolerant storage to persistent media such as a computer hard drive. Function libraries allow developers to concentrate specifically on the wider purpose of the mobile agents in their system rather than on their basic execution requirements.

Most agent servers make extensive use of multi-threading in order to execute agent activity. Mobile agents usually run in an independent thread, allowing multiple agents to execute concurrently. Mobile agent developers can also make use of multithreading within individual agents to allow simultaneous activity to be executed e.g. agent communication while also conducting high level planning etc. Mobile agents can make use of many communication mechanisms including sockets, RMI and CORBA. Although remote communication using these mechanisms is possible, migration and local communication are more often employed in mobile agent systems because this helps to reduce network load in comparison to other distributed computing paradigms which must all communicate remotely across networks.

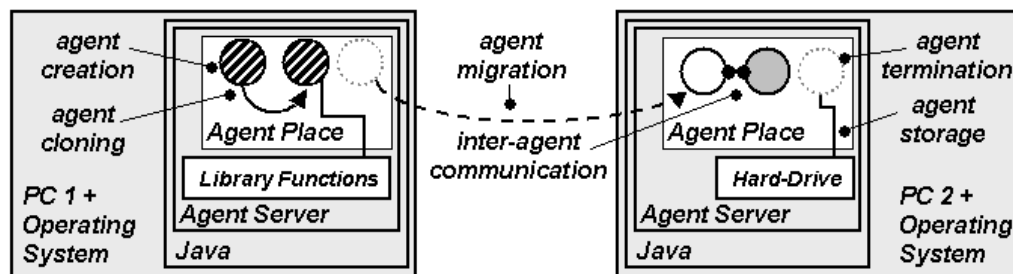


Fig. 3 Mobile agent execution environment

3.2 Characteristics and functionality

Mobile agents are goal driven and autonomous, they can be reactive, adaptable, dynamic, temporally continuous, operate asynchronously, communicate, and learn. Because they can provide all of these characteristics and also move from place to place, mobile agents can provide the functionality found in all other distributed computing architectures combined. In addition they provide a more natural design philosophy for distributed computing systems when compared to other distributed computing paradigms i.e. traditional client/server based architectures require more complex interconnected networks of heterogeneous computing nodes than mobile agents. Using mobile agents we do not need to consider network connections and consider all distributed computing nodes as offering equivalent functionality. This allows the development of systems which are more likely to be logically sound/robust.

Mobile agents can be used to produce intelligent, dynamic, fault tolerant and scalable distributed systems and reduce network load. This has allowed them to be applied in a range of application areas (5) including, computational outsourcing, dynamic network management, load balancing, personalised user presence, software deployment, intelligent data, temporary applications, application of dynamic protocols and intelligent remote action (6).

4 IMPLEMENTATION

This section describes how we have implemented ALLIANCE in a mobile agent environment, starting initially with a description of the hardware/software employed.

4.1 Hardware/software

Our research hardware/software implementation is shown in Fig. 4. It includes two PCs, one running Windows^{XP} and the other Linux. The PC-Windows^{XP} contains Java 1.4, Grasshopper 2 (GH2) Agent Platform (7) (to host mobile agents), a Timing Server (to time experiments)

and Apache Web Server v2.0 (to serve Java mobile agent .class files to GH2). The PC-Linux contains SSH (to allow remote login to multiple robots), SFTP (to allow downloading of files to multiple robots), and a Timing Server. The implementation also includes three ActivMedia Pioneer 2DX mobile robots (8) each running Linux and network accessible via wireless LAN. Each robot contains Java 1.3, GH2 and ActivMedia;fs Ai a robot contr d software

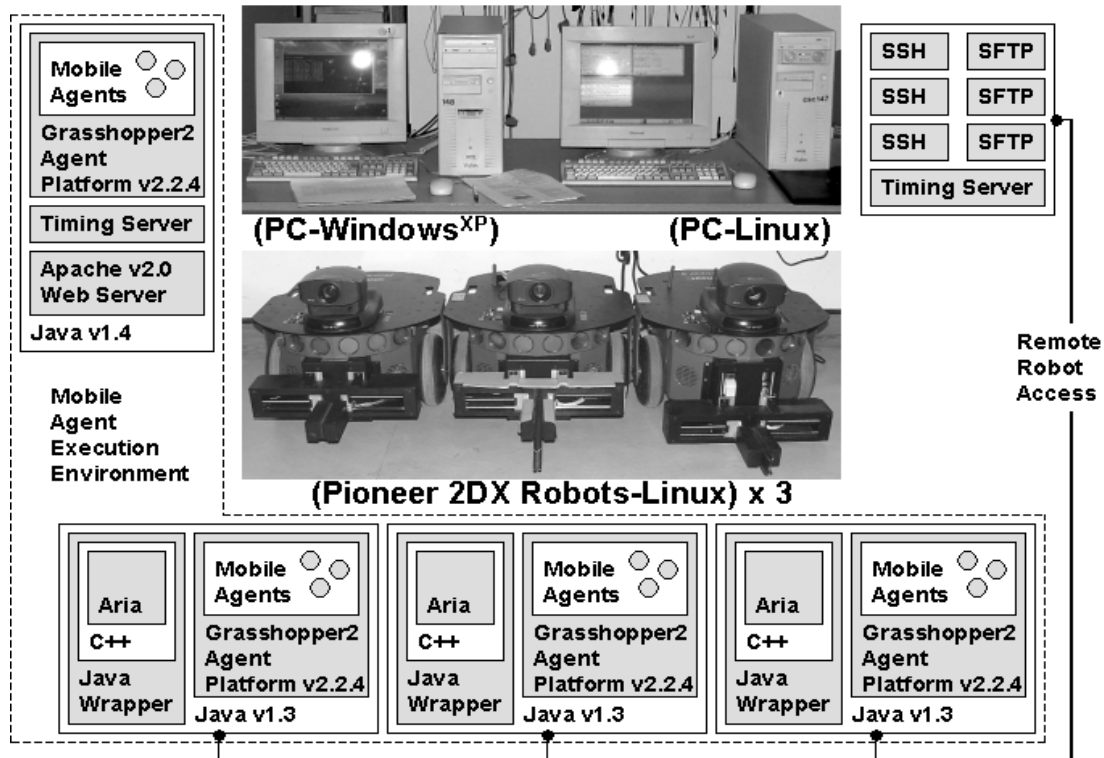


Fig. 4 Mobile robot/mobile agent implementation

GH2 is a free OMG MASIF compliant Java based mobile agent server which 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;fs r-time behaviour. Apache Web Server is used to store agent Java .class files. The GH2 agent servers obtain .class files from this central repository.

ActivMedia;fs Ai a robot contr d software allows various levels of contr d to be executed on ActivMedia Pioneer 2DX robots, it is written in C++ but provided with a Java Wrapper which allows Aria commands to be written directly in our Java based mobile agents. The Timing Servers are multi-threaded Java based servers which listen on known ports and register 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.

4.2 Architecture

Fig. 5 shows our ALLIANCE implementation with the main element of an ALLIANCE Control Agent (ACA), a mobile agent which engages in three main activities: Calculation of MB;fs, execution of BS;fs, and IC. To conduct IC, calculate NB;fs and execute BS

concurrently, we implement these activities as subclasses of the thread class. We start each thread in the `ACA::live()` function allowing the mt to execute simultaneously. Using synchronised methods these classes are able to access shared variables within the main body of the ACA allowing coordinated behaviour. As these activities operate independently the main agent thread is free to interact with the server and other agents. The MB thread calculates a motivation value for each BS every 100ms. The BS contains code for the execution of any BS selected by MB in the MB thread and executes every 100 ms.

For IC the IC thread creates and interacts with a second form of mobile agent, an ALLIANCE Communication Agent (AComm). At each cycle of the IC thread i.e. every 4 seconds the IC thread on each robot creates a single AComm. This agent is created with a list of all known robot addresses, the ID of the creating robot and the behaviour of the creating robot. Upon creation this agent automatically creates a copy of itself on all the other robots in its list and then removes itself from the creating robot.

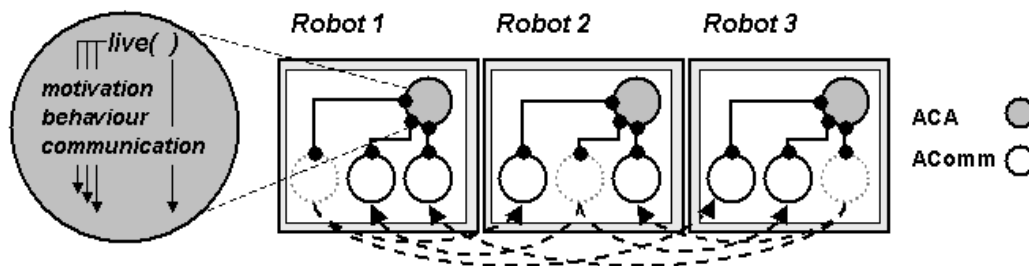


Fig. 5 Implementation architecture

After creating the AComm the IC thread uses the agent server to locate and interact locally with any AComm which have been copied to his robot from other robots. It obtains the ID information and current behaviour of any AComm currently residing on the robot, thereby obtaining information on the activity of all other communicating teammates. After providing the local IC thread with their information payload AComms remove themselves from the robot host in order to prevent a build-up of agent numbers and resource use. The IC conducted in this way allows the MB thread to be provided with relevant information with which to calculate motivation values for behaviour selection.

5 EXPERIMENTS

Using the implementation shown in Fig. 4 and 5, two sets of experiments were conducted. These are described in the following sections.

5.1 Adaptability experiments

Adaptability experiments were designed to investigate how mobile agents might affect the adaptability characteristics (i.e. the ability to update or adapt system code at run-time) of an ALLIANCE controlled multiple robot system. One set-up used an ALLIANCE mobile agent implementation (in which mobile agents are employed to allow existing ACAs to be dynamically updated by new ACAs at run-time), whilst the second investigated the updating of a traditional static code ALLIANCE implementation (using remote robot login (SSH) and file transfer using (SFTP)).

Fig. 6a shows mobile agent adaptability experiment activities. Initially existing ACA_i's are used to control multiple robots via Robot Agents (RA_i's) (1). A GH2 platform is then started which automatically creates a new ACA (2). The new ACA clones itself to all robots, at which clones communicate with existing ACA_i's causing them to move the robot from their robot (3). The new ACA_i's then connect to the robot via RA_i before communicating with a timing server (4). Timing of an experiment trial begins on the creation of the GH2 platform in step (2) and ends when the last new ACA communicates with the timing server in step (4).

Fig. 6b shows static code adaptability experiment activities. Initially existing Alliance Control Programs (ACP) are used to control multiple Robots (1). SFTP is then used to download new ACP_i's (2) and SSH used to stop existing ACP_i's (3). SSH is then used to start the new ACP_i's which connect to their robot and communicate with a timing server (4). Timing of an experiment trial begins on the creation of the first SFTP connection to robots in step (2) and ends when the last new ACP communicates with the timing server in step (4).

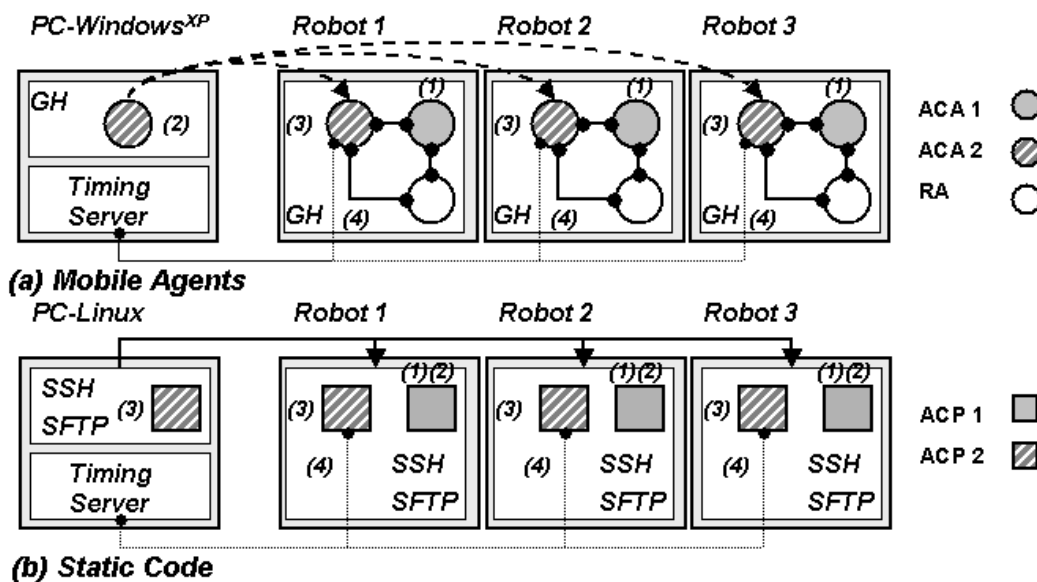


Fig. 6 Adaptability experiments

5.2 Fault tolerance experiments

Fault tolerance experiments were designed to investigate how mobile agents might affect the fault tolerance characteristics (i.e. the ability of a system to remain operational in the presence of faults) of an ALLIANCE controlled multiple robot system. One set-up used an ALLIANCE mobile agent implementation (in which an existing ACA 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 replacement robot), whilst the second investigated a traditional static code ALLIANCE implementation (in which a failing robot loses mission level data when it fails and is replaced using a new robot with no previous mission experience).

Fig. 7a shows mobile agent fault tolerance experiment activities. Initially an ACA executing on Robot 1 detects a terminal fault and communicates with a timing server (1). This ACA then migrates to Robot 2 (2) and waits until a Robot Start Up Agent (RSUA) communicates with it to show that a new Robot 1 has been started (3). The ACA waiting on Robot 2 then migrates to the New Robot 1 connects to the robot via an RA and communicates with the

timing server once more (4). Timing of an experiment trial begins on the connection of ACA to the timing server in step (1) and ends on ACA connection to the timing server in step (4).

Fig. 7b shows static code fault tolerance experiment activities. Initially an ACP executing on Robot 1 detects a terminal fault and communicates with a timing server (1). SFTP is then used to download a new ACP to a new Robot 1 (2). SSH is then used to start this new ACP (3). The new ACP connects to the new Robot 1 and then communicates with the timing server (4). Timing of an experiment trial begins on the connection of ACP to timing server in step (1) and ends on ACP connection to the timing server in step (4).

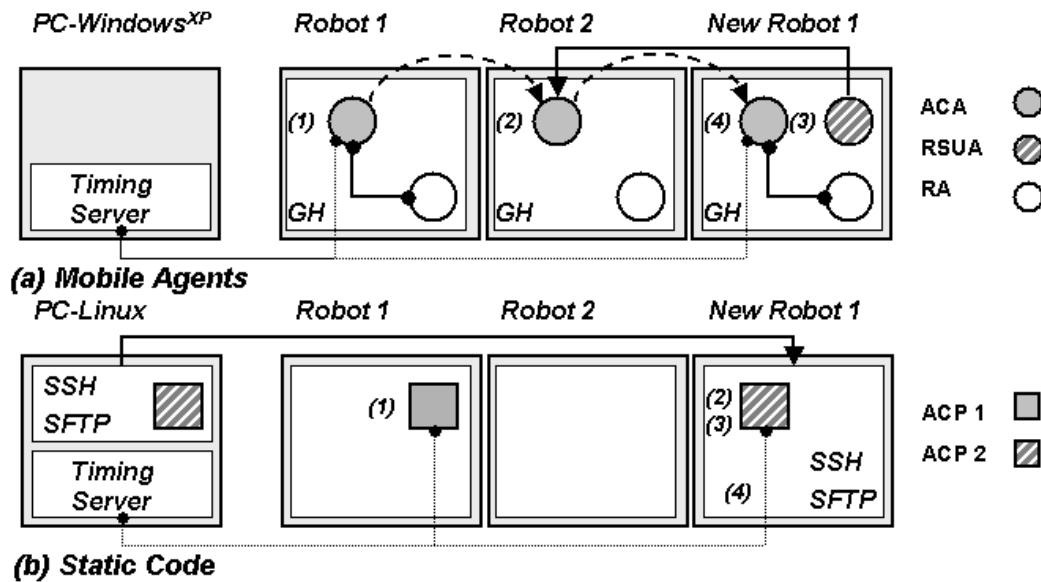


Fig. 7 Fault tolerance experiments

6 EXPERIMENTAL RESULTS AND ANALYSIS

6.1 Adaptability experiments

Adaptability experiment results are shown in Fig. 8 and Table 1. The adaptability experiments show that it is considerably quicker and easier to update and adapt the control software on multiple robots using mobile agents in comparison to a traditional method using static code. In the experiments scripts were used wherever possible in the static code experiments to remove human performance factors and to speed up and automate the updating process. Despite this mobile agents updated code much more quickly.

In addition the times for mobile agents include the time required to start an agent server at the PC-Windows^{XP}, while in practice such a server would normally be in operation at all times. The mobile agent performance is an improvement over traditional methods because agent creation occurs in parallel on all robots, while in the static code implementation activities occur in series. Due to this parallel execution, as robot team numbers increase, the mobile agent to static code time ratio should widen i.e. mobile agents relative to static code should be ever more effective as robot number increases.

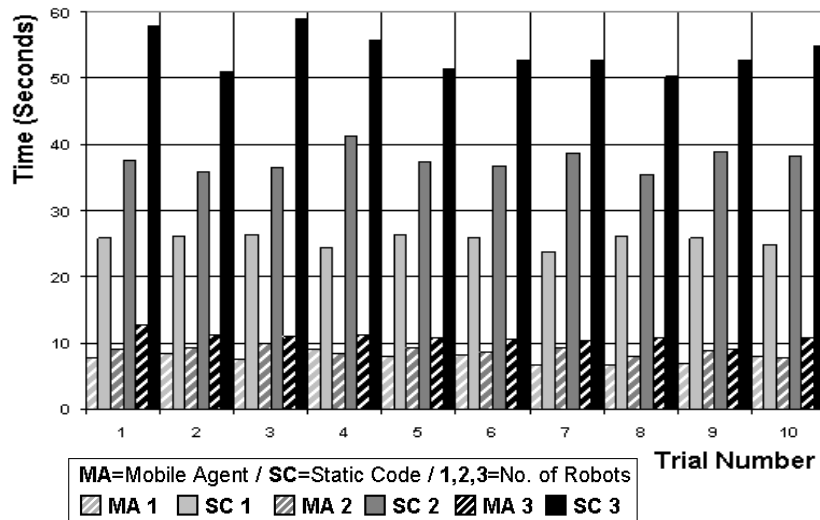


Fig. 8 Adaptability experiment results chart

Table 1: Adaptability experiment results table

Type\Trial	1	2	3	4	5	6	7	8	9	10
MA 1	7.8	8.4	7.6	8.9	8.0	8.1	6.6	6.6	6.9	7.8
SC 1	25.9	26.1	26.4	24.3	26.2	26.0	23.7	26.2	25.8	24.8
MA 2	9.0	9.2	10.0	8.4	9.2	8.5	9.2	8.0	8.7	7.8
SC 2	37.5	35.8	36.5	41.3	37.4	36.8	38.6	35.4	38.8	38.2
MA 3	12.6	11.2	11.0	11.3	10.9	10.5	10.4	10.8	8.9	10.7
SC 3	58.0	51.0	59.0	55.8	51.3	52.7	52.7	50.2	52.6	54.8

6.2 Fault tolerance experiments

Fault tolerance experiment results are shown in Fig. 9 and Table 2. The fault tolerance experiments have shown that using mobile agents it is possible to retain the state and data of a failing robot. However it took more time (approximately 7-8 seconds more) using mobile agents to start up a new replacement robot in comparison to the static code implementation. Although this process took longer using mobile agents, the traditional static code implementation lost robot state and data when the robot failed.

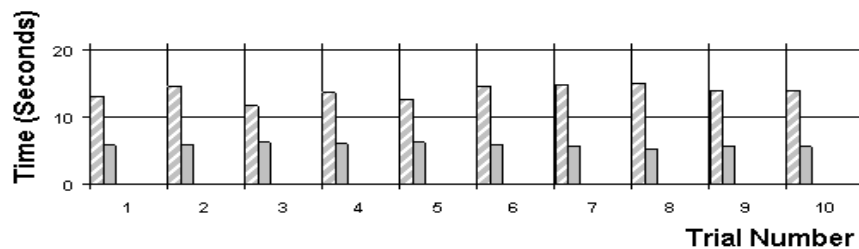


Fig. 9 Fault tolerance experiment results chart

Table 2: Fault tolerance experiment results table

Type\Trial	1	2	3	4	5	6	7	8	9	10
MA 1	13.0	14.5	11.7	13.6	12.6	14.7	14.8	15.0	13.9	13.9
SC 1	5.8	5.8	6.3	6.0	6.2	5.9	5.7	5.3	5.7	5.6

7 CONCLUSIONS AND FUTURE WORK

In this paper we have examined an implementation of the ALLIANCE architecture in networked robots using mobile agents in which we extend adaptability and fault tolerance. These functional extensions arise as a result of the implementation of ALLIANCE in the mobile agent environment and not through its architectural adaptation. These extensions could equally be achieved with other distributed mobile robot control architectures. Mobile agents offer many opportunities for the extension of functionality in multiple robot systems, because they unify the functionality found in all other distributed systems, and provide a natural design philosophy for distributed systems such as multiple robot architectures (based upon an equal allocation of basic functionality throughout all mobile agent server enabled system components). In conjunction these characteristics can allow multiple robot system developers to employ mobile agents as a tool to develop robust cohesive architectures in which system structure can be easily extended by making use of in built functionality available in the mobile agent environment. We are using this implementation of ALLIANCE as the control selection mechanism of a multiple tele/autonomous robot architecture (9) in which we aim to make further use of the beneficial functionality which mobile agents offer.

REFERENCES

- [1]. Parker L. E. (1994), "ALLIANCE: An Architecture for Fault Tolerant, Cooperative Control of Heterogeneous Mobile Robots", Proc. of the IEEE RSJ/ GIJt. Conference on Intelligent Robots and Systems (IROS '94), September 1994, pages 776-783.
- [2]. Parker L. E. (2001), "Evaluating Success in Autonomous Multi-Robot Teams: Experiences from ALLIANCE Architecture Implementations", Journal of Theoretical and Experimental Artificial Intelligence, 13, 2001, pages 95-98.
- [3]. Yu L., Tsui P.W., Zhou Q. and H. Hu (2001), "A Web-based Tele-robotic System for Research and Education at Essex", Proc. of IEEE ASME International Conference on Advanced Intelligent Mechatronics, Como, Italy, 8-11 July 2001, pages 284-289.
- [4]. Tsui P.W. and Hu. H (2002), "A Framework for Multi-Robot Foraging over the Internet", Proceedings of IEEE International Conference on Industrial Technology 2002
- [5]. Milojicic D. et al., "Mobile Agent Applications", IEEE Concurrency, Ju-September 1999, pages 80-90.
- [6]. Camarinha-Matos L. and Vieira W., "Intelligent Mobile Agents in Elderly Care", Journal of Robotics and Autonomous Systems 27 (1999), pages 59-75.
- [7]. IKV++ (2003), "Grasshopper 2 Homepage", World Wide Web <http://www.grasshopper.de/>, IKV++ Technologies AG, 2003.
- [8]. ActivMedia (2003), "Software Documentation and Technical Support", World Wide Web, <http://robots.activmedia.com/techhome.html>, 2003.
- [9]. Cragg L. and Hu, H. (2003), "Application of MAS to Robust Tele-operation of Internet Robots in Nuclear Decommissioning", Proc. of IEEE International Conference on Industrial Technology, Maribor, Slovenia, 10-12 December 2003, pages 1214-1219.