

IMPLEMENTING ALLIANCE IN NETWORKED ROBOTS USING MOBILE AGENTS

Liam Cragg and Huosheng Hu

*Department of Computer Science, University of Essex,
Wivenhoe Park, Colchester C04 3SQ, United Kingdom
E-mail: {lmcrag, hhu}@essex.ac.uk*

Abstract: Mobile Agents (MA's) combine the functionality found in all other distributed computing paradigms in a unified environment and provide a natural design philosophy for distributed computing systems such as multiple robot architectures. MA's offer a wide range of positive functional characteristics. In this paper we present two examples to show how such characteristics can be used to extend the adaptability and fault tolerance of a multiple robot architecture. We do this through a mobile agent implementation of the ALLIANCE architecture in multiple networked robots.

Copyright © 2004 IFAC

Keywords: Networked Multi-Robots, Mobile Agents, Fault Tolerance, ALLIANCE Architecture.

1. INTRODUCTION

The ALLIANCE architecture was developed for fault tolerant control in multiple robot systems (Parker, 2001). While ALLIANCE has been shown to be effective for this purpose, real-world applications continue to demand advances in fault-tolerant architectures. In its original implementation ALLIANCE used one-way radio frequency message passing for inter-robot communication between static code residing on multiple robots (Parker, 1994).

Networked robots and modern distributed computing paradigms offer us opportunities to extend fault tolerant multiple robot architectures like ALLIANCE by making use of Internet protocols and hardware for communication and provide new functionality for developing distributed systems. One such modern distributed computing paradigm is MA's. MA's provide the functionality found in all distributed computing paradigms in a unified mechanism which also provides a natural design philosophy for the development of distributed computing systems like multiple robot architectures (Cragg and Hu, 2003).

MA's have a number of positive characteristics including fault tolerance, heterogeneity, autonomy, intelligence, reactivity, adaptability, scalability and asynchronous execution, which can be used to extend the functionality available in distributed systems. In this paper we show how MA's can be used to extend the functionality found in distributed multiple robot architectures. We achieve this specifically through

the implementation of the ALLIANCE architecture in a MA environment.

The rest of this paper is organised as follows. Section 2 provides background information on the ALLIANCE architecture, networked robots and distributed computing paradigms. Section 3 provides a detailed description of the MA paradigm. Section 4 describes our implementation of ALLIANCE in a MA environment. Section 5 shows examples of the extension of ALLIANCE functionality which a MA implementation provides. Section 6 presents some conclusions and discusses future work.

2. BACKGROUND

This section provides some background information relating to the ALLIANCE architecture, networked robots, and MA's.

2.1 The ALLIANCE Architecture

A simplified diagram of the ALLIANCE architecture can be seen in Fig. 1. The main components of ALLIANCE are Motivational Behaviours (MB's) and Behaviour Sets (BS's). MB's receive input from inter-robot communication (IC) and sensors, and are linked to an associated BS. A BS receives input from sensors and provides output to robot actuators controlling the behaviour of a robot at runtime. Each BS relates to an independent activity (which may contain sub-tasks).

Each MB determines whether its associated BS is selected at runtime by calculating the robots motivation to perform it. Motivation in ALLIANCE 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. A more detailed description of ALLIANCE and the calculation of motivation can be found in (Parker, 1994).

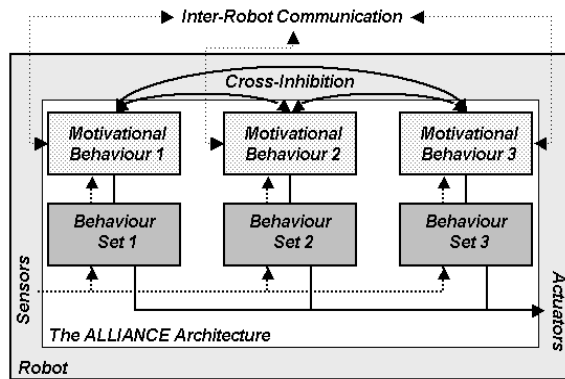


Fig. 1 The ALLIANCE Architecture

2.2 Networked Robots

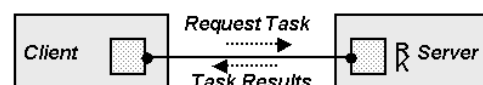
Networked robots use distributed computing hardware and software to allow communication between robot team members. Such systems often employ Internet protocols such as TCP/IP and UDP/IP for communication and wireless LAN as transmission hardware. Networked robot systems exist which represent the spectrum of robot systems from single tele-operated robots e.g. University of Essex Tele-robot (Yu, et al., 2001) to multiple autonomous robot systems e.g. MURDOCH (Gerkey and Mataric, 2002). The unifying characteristic of these systems is their use of networking technology for communication. In the case of tele-operated Internet robots as in (Yu, et al., 2001) this makes their functionality widely accessible and cost effective.

2.3 Distributed Computing Paradigms

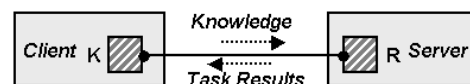
A number of distributed computing paradigms exist with which distributed computing systems such as multiple robot systems can be structured. These include:

- *Client/Server*: In the client/server 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.
- *Remote Computation*: In the remote computation paradigm (Fig. 2b) a client has knowledge while a server contains 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 then uses the client's knowledge and its own resources to perform the task before returning a result to the client.

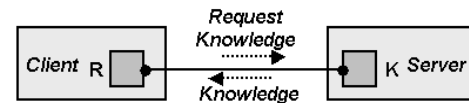
- *Code on Demand*: In the code on demand paradigm (Fig. 2c) a client has resources while a server contains knowledge to perform a task. For the client to obtain a result to a task, it must send a message to the server to request knowledge. The client uses the server's knowledge and its own resources to perform the task and obtain a result.
- *MA's*: In the MA paradigm (Fig. 2d) an agent server is host to a 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 MA moves to the new agent server to access the resource. When the MA has completed its task it can remain at the new server or move to another agent server carrying its knowledge and task results.



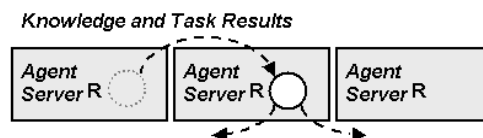
a) Client/Server



(b) Remote Computation



(c) Code on Demand



(d) MA

Fig. 2 Distributed Computing Architectures

3. MOBILE AGENTS

In the previous section we compared activity in a variety of distributed computing paradigms. In this section we specifically examine the MA paradigm in more detail.

Execution Environment

MA's exist within an execution environment, provided by multiple agent servers (as shown in 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 MA's are written in Java. Because Java is an interpreted computer language, the same Java code can execute on a variety of heterogeneous computer platforms. This

means that Java based MA's can also execute in heterogeneous computing environments.

Because Java is interpreted it allows agent server developers to employ Java based and bespoke security mechanisms to prevent the execution of malicious agents in an agent place thereby protecting host computers.

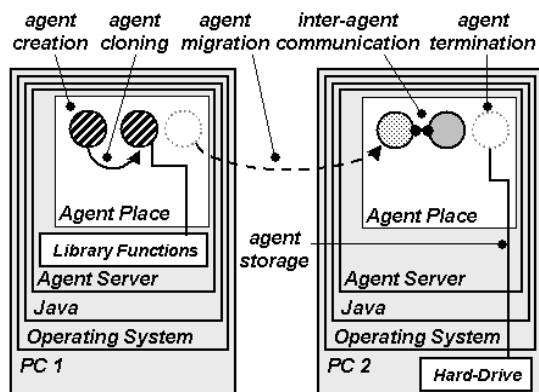


Fig. 3 MA Execution Environment

Multiple host computers can provide a distributed (but unified) MA execution environment each hosting a single agent server which can be uniquely identified using the host IP address. Agent servers may contain multiple places, between which multiple MA's can simultaneously migrate.

Basic MA's are lightweight computing components, because many of the functions which they execute e.g. migration, cloning, etc are provided by agent servers in a function library. Agent servers provide MA's with a range of functionality, including creation, destruction, cloning, migration and fault tolerant storage to persistent media such as a computers hard drive.

This means that developers of systems which employ MA's can concentrate specifically on the wider purpose of the MA's within their system.

Most agent servers make extensive use of multi-threading in order to execute agent activity. MA's usually run in an independent thread, allowing multiple agents to execute concurrently. MA developers can also make use of multithreading within individual agents to allow simultaneous activity to be executed e.g. agent communication while simultaneously conducting high level planning etc.

MA's can make use of many communication mechanisms including sockets, RMI and CORBA. Although remote communication using these communication mechanisms is possible, migration and local communication are more often employed in MA systems because this helps to reduce network communication load in comparison to all other distributed computing paradigms which must communicate remotely across a network.

Characteristics and Functionality

MA's are goal driven and autonomous, they can be reactive, adaptable, dynamic, temporally continuous, operate asynchronously, communicate, and learn. Because they can provide some or all of these characteristics and also move from place to place, MA's can provide the functionality found in all other distributed computing architectures combined. They can be used to provide intelligent, dynamic, fault tolerant and scalable distributed systems and reduce network communication. This has allowed them to be applied effectively in a number of application areas (Milojicic, et al., 1999) 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.

4. IMPLEMENTATION

Having provided some detailed background on MA's the next section describes how we have implemented ALLIANCE in a MA environment.

4.1 Hardware/Software

We have implemented ALLIANCE in the MA hardware/software shown in Fig. 4. This environment contains a number of networked PCs each running Windows^{XP} operating system and Java 1.4. Grasshopper 2 Agent Platform (IKV++, 2004) is adopted as our agent development environment and ActivMedia's Pioneer Simulator (ActivMedia, 2004) is used to provide simulated Pioneer 2DX robot functionality allowing us to subsequently transfer the architecture to real Pioneer 2DX's in our laboratory.

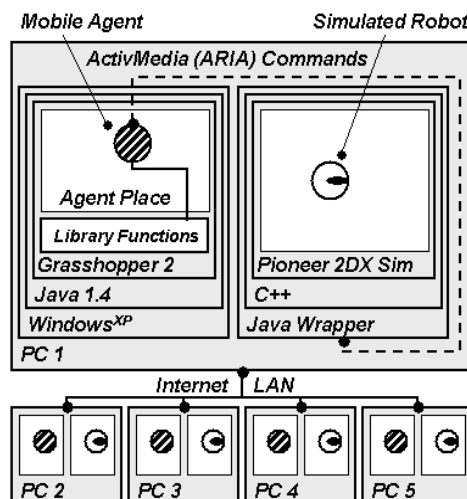


Fig. 4 Hardware/Software Implementation

Grasshopper is OMG MASIF compliant written in Java and provides a number of useful library functions for the creation and manipulation of mobile (or stationary) agents.

The main body of an agent is the live() method which is executed by Grasshopper as an independent thread. A developer overrides the live() method and places their code within it in order to determine the behaviour of the agent at run time.

4.2 Architecture

Fig. 5 shows our ALLIANCE implementation with the main element of an ALLIANCE Control Agent (ACA), which engages in three main activities: Calculation of MB's, execution of BS's, and performing IC.

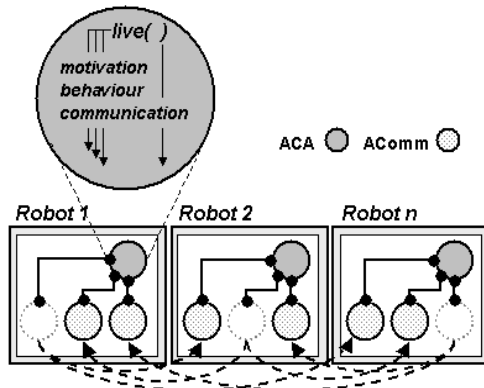


Fig. 5 Implementation Architecture

To conduct IC, calculate MB's and execute BS's concurrently, we implement these as subclasses of the thread class. We start each thread in our ACA's live() method allowing them to execute simultaneously. As these activities are operating independently of the main agent thread it is free to interact with the agent server and other agents.

The MB thread calculates motivation for each BS every 100ms. The BS thread contains code for the execution of the BS selected by MB's in the MB thread.

For IC the IC thread creates and interacts with a second form of MA, 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 the addresses of all other known robot places, 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 of addresses and then removes itself from the creating robot place.

After creating the AComm the IC thread uses the agent server to locate and interact locally with any AComms which have been copied to this robot from other robots. It obtains the ID information and current behaviour of any AComms currently residing on the robot, thereby obtaining information on the activity of all other communicating team-mates.

After providing the local IC thread with their information payload AComms remove themselves from the robot. The IC conducted in this way allows

the MB thread to be provided with relevant information with which to calculate BS motivation values.

5. EXPERIMENTS

Having described how ALLIANCE is integrated in a MA environment we now provide two examples to show functionality this implementation can provide:

5.1 Adaptability

The aim of these experiments was to examine the adaptability characteristics of ALLIANCE embedded in a MA environment relative to a more traditional implementation by updating the control software of a multiple robot architecture.

MA Implementation

Using ALLIANCE embedded in MA's we conducted the activities shown in Fig. 6 which represent a single experimental trial.

After ACA's are created on a number of PC's each controlling a simulated robot, a new ACA is created on an external PC (1). This automatically clones itself simultaneously to the PC's on which the existing ACA's are located (2). On reaching their destination the new ACA's communicate with the existing ACA's (3). This causes the existing ACA's to stop controlling their robot (4), after which the new ACA's are able to take over control of the robot (5). These experiments were conducted on robot teams ranging in number from 1-5. The total time taken from new ACA creation (1) to control of the simulated robot by the new ACA (5) was recorded. The results of 5 experimental trials for robot teams ranging in number from 1-5 are shown in Table 1 and Fig. 7.

Traditional (T) Implementation

In a traditional implementation, ALLIANCE control code would be constructed in a stationary application. If we wished to update the control code of a robot we would need to remotely log in to the robot, stop its existing code, manually download new code and remotely execute the new control code.

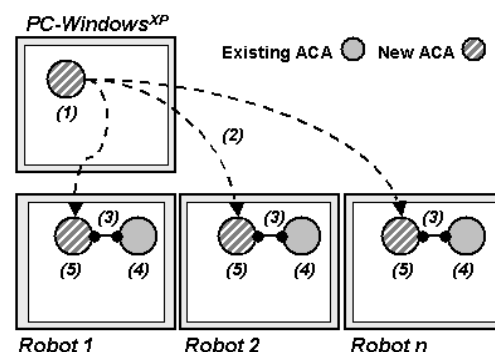


Fig. 6 State Changes in MA Adaptability Experiment

Table 1 Adaptability Results (Time in Seconds)

Trial No.	1	2	3	4	5
MA 1 Robot	6.19	6.12	6.09	6.06	6.03
MA 2 Robots	6.28	6.06	6.43	6.07	6.22
MA 3 Robots	6.40	6.29	6.28	5.97	6.22
MA 4 Robots	6.37	6.10	6.25	6.57	6.28
MA 5 Robots	6.37	6.19	6.22	6.19	6.00
T1 Robot	21.66	26.53	22.15	26.09	21.44
T2 Robots	59.09	55.59	73.22	60.53	61.34
T3 Robots	96.50	84.53	89.15	93.22	79.19
T4 Robots	108.90	120.66	109.71	108.47	116.87
T5 Robots	157.68	165.06	160.59	129.06	139.40

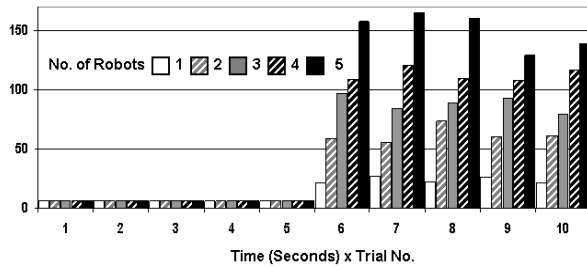


Fig. 7 MA (1-5) T (6-10) Adaptability Results

Due to the need to maintain network security, remote download and execution functionality between Windows^{XP} PC's in our laboratory is prohibited preventing us from employing this functionality in our simulated experiments (although it is available on our real robots). Therefore in this set of experiments we rely on MA's to simulate the functionality we require. This is achieved by conducting the following activities.

After ACA's are created on a number of PCs each controlling a simulated robot, a new ACA is created on an external PC (1). This is manually cloned to one robot to simulate the downloading of a static application to a remote robot (2). A message was then sent from an additional agent constructed at the external PC to communicate remotely with the existing ACA, and the new ACA (in order to simulate remote log in of a real robot). The agent on the external PC was used to stop the existing ACA (3), and start the new ACA (4). This process was then repeated for each subsequent member of the robot team (5).

These experiments were conducted on robot teams ranging in number from 1-5. The total time taken from new ACA creation (1) to control of the simulated robot by all new ACA's (5) was recorded. The results of 5 experimental trials for robot teams ranging in number from 1-5 are shown in Table 1 and in number 6-10 in Fig. 7.

These results show that the MA implementation can be updated in approximately 6 seconds, which appears to remain constant despite increases in team size due to the parallel execution of MA's on robots. In a traditional implementation 21 to 165 seconds is required for similar activity. Adaptation time increases with robot number in the traditional implementation due to the requirement to adapt robots sequentially.

5.2 Fault Tolerance

The aim of these experiments was to examine the fault tolerance characteristics of ALLIANCE embedded in a MA environment relative to a more traditional implementation through the retention of control software in a multiple robot architecture.

MA Implementation

Using ALLIANCE embedded in MA's we conducted the activities shown in Fig. 8 which represent a single experimental trial.

After ACA's are created on a number of PC's each controlling a simulated robot, we then simulated the terminal failure of one of the robots (1). This causes this robot's ACA to clone itself simultaneously to other robot team members (2). The clones await the introduction to the team of a new robot (3) which announces its arrival by sending a message to the existing robot team members (4). The cloned ACA's then migrate to the new robot (5) and the first agent to arrive takes control of the robot (6).

These experiments were conducted on robot teams in which a single robot failed. The total time taken from ACA cloning (1) to control of the newly added simulated robot (6) was recorded. The results of 5 experimental trials for this activity are shown in Table 2 and Fig. 9.

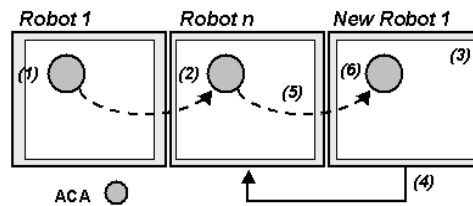


Fig. 8 State Change in MA Fault Tolerance Experiment

Traditional (T) Implementation

In a traditional implementation, ALLIANCE control code would be constructed in a stationary application. This means that if a robot had a terminal failure ALLIANCE control code would be lost. However we could add a new robot to the team in order to replace it, manually downloading new ALLIANCE control code and remotely initiating its execution. As previously mentioned our experimental environment prevents us from conducting remote download and execution, therefore in this set of experiments we again rely on MA's to simulate the functionality we require.

After ACA's are created on a number of PC's each controlling a simulated robot one of which was experiencing a simulated failure (1), a new ACA was created on an external PC (2). This was manually cloned to a new robot to simulate the downloading of a static application to a new robot team member (3).

Table 2 Fault Tolerance Results (in Seconds)

Trial No.	1	2	3	4	5
MA 1 Robot	12.47	11.84	10.28	11.34	10.53
T 1 Robot	11.47	11.87	11.16	11.15	10.88

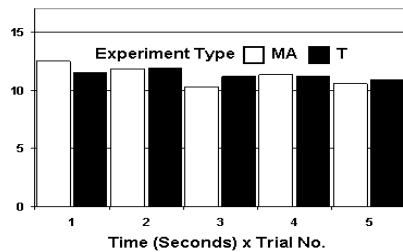


Fig. 9 Fault Tolerance Results

A message was then sent from an additional agent constructed at the external PC to communicate remotely with the new ACA (in order to simulate remote log in of a real robot). The agent on the external PC was used to start the new ACA (4). These experiments were conducted on robot teams in which a single robot failed. The total time taken from ACA creation (2) to control of the newly added simulated robot (4) was recorded. The results of 5 experimental trials for this activity are shown in Table 2 and Fig. 9.

The results show that the control code of a robot in our experiments can be retained by migrating it to other robots and then to a new robot in approximately the same time as it takes to download new control code to a new robot team member using a more traditional implementation i.e. 12 seconds.

5.3 Discussion

The results of our experiments show that there is an advantage to the use of MA's for updating multiple robot control code, due to the autonomous parallel execution of MA's relative to a more traditional implementation. This functionality allows control code in a robot team to be updated much more rapidly. This will allow system development to be conducted more quickly as new control algorithms can be more easily downloaded and tested, and will allow an active multiple robot team to be more easily adapted at run time.

In addition the fault tolerance experiments show that it is possible to retain the control code and data of an existing robot without incurring an overhead in terms of time taken to achieve this activity. The process of retaining existing robot control code and data in our experiments taking approximately the same time as that required to install a new robot team member with new control code having lost mission level data in a failing robot.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have examined an implementation of the ALLIANCE architecture in networked robots using MA's in which we extended adaptability and fault tolerance.

These functional extensions arise as a result of the implementation of ALLIANCE in the MA environment and not through its architectural adaptation. These extensions could equally be achieved with other distributed mobile robot control architectures. MA's offer many opportunities for the extension of functionality in multiple robot systems, but their main advantage arises because of their unification of functionality found in all other distributed computing paradigms, in conjunction with a natural design philosophy based upon an equal allocation of basic functionality throughout all mobile agent server enabled system components (i.e. in order to achieve the same functionality without mobile agents a more complex disparate assembly of components from alternative distributed computing paradigms would be required).

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 functionality can be easily extended later by making use of inbuilt functionality available in the mobile agent environment.

REFERENCES

- ActivMedia (2004), "Software, Documentation and Technical Support", *ActivMedia Robotic*, <http://robots.activmedia.com/>, 2004.
- Cragg L. and Hu H. (2003), "Application of MA's to Robust Tele-Operation of Internet Robots in Nuclear Decommissioning", *Proceedings of IEEE International Conference on Industrial Technology-ICIT03*, p:1214-1219, Dec 2003.
- Gerkey B.P. and Mataric M.J. (2002), "Sold!: Auction methods for multi-robot coordination", *IEEE Transactions on Robotics and Automation, Special Issue on Multi-robot Systems*, 18(5):758-768, 2002.
- IKV++ (2004), "Grasshopper 2 Homepage", *IKV++ Technologies AG World Wide Web*, <http://www.grasshopper.de/>, 2004.
- Milojicic D. et al. (1999), "Mobile Agent Applications", *IEEE Concurrency*, July-Sept 1999, 80-90.
- Parker L.E. (1994), "ALLIANCE: An Architecture for Fault Tolerant, Cooperative Control of Heterogeneous Mobile Robots", *Proceedings of the IEEE/RSJ/GI Int. Conference on Intelligent Robots and Systems (IROS '94)*, September 1994: 776-783.
- 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: 95-98.
- Yu L., Tsui P.W., Zhou Q. and H. Hu (2001), "A Web-based Tele-robotic System for Research and Education at Essex", *Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, p:284-289, Como, Italy, 8-11 July 2001.