Agent Replication in Multi-Agent Systems

Alan Fedoruk
Department of Computer Science
University of Saskatchewan
57 Campus Drive
Saskatoon, SK S7N 5A9
Canada
email: amf673@mail.usask.ca

Supervisor: Prof. Ralph Deters

Abstract

The complex nature of multi-agent systems makes them vulnerable to faults and system failures. This paper examines replication of individual agents within a multi-agent system as a means of increasing agent fault tolerance and lowering the overall system failure rate. Once an agent is replicated, inter-agent communications, read and write consistency, resource locking and results synthesis need to be managed. This paper defines agent replication and looks at ways that the issues agent replication raises can be addressed. The results of an experiment which simulates a multi-agent system using replicated agents are presented.

1 Introduction

The goal of this research is to determine if there are effective methods of replicating agents in a multi-agent system to increase the fault tolerance of an individual agent and lower the overall system failure rate.

Multi-agent systems (MASs) are composed of interacting, autonomous and distributed software agents, situated in a dynamic environment. These characteristics mean that at run-time an MAS is non-deterministic, which makes it susceptible to any of various types of faults. As it is not possible at design time to foresee all possible states that may occur when one part of the system experiences a fault, faults can lead to overall system failure.

Replication of one or more of the individual agents within an MAS provides redundant copies of the agent, increases the tolerance of the agent to certain types of faults and hence lowers the overall system failure rate. The focus of this paper is to define what agent replication is, identify issues associated with agent replication and propose methods of dealing with those issues.

This paper is structured as follows. Section 2 describes multi-agent systems and their properties and describes fault and failures as they relate to agent replication. Section 3 presents agent replication, describing what it can be used for, the problems that it introduces and some methods for dealing with those problems. Section 4 describes a simulation that was performed. Section 5 reviews related literature. Section 6 and 7 identify future work and present conclusions.

2 Multi-Agent Systems, Faults and Failures

There are four essential characteristics of a multi-agent system: an MAS is composed of autonomous software agents, an MAS has no single point of control, an MAS interacts with a dynamic environment, and the agents within an MAS are social (agents interact with each other and may form relationships).

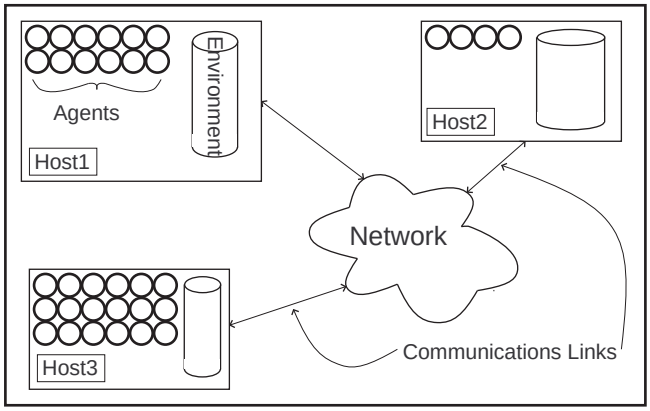

Figure 1: A Multi-Agent System

As described in [Wooldridge 2000], the main sources of complexity in an MAS are the agent's environment, the nature of the interaction between the agent and the environment and the nature of the task the agent is performing. Additional complexity is introduced by interactions between the agents within the system and by the distributed nature of MASs. In this paper, complexity refers to the difficulty of implementing computer systems, not mathematical complexity. [Wooldridge 2000] As more agents are added to an MAS, all five sources of complexity increase. This complexity makes MASs unpredictable and susceptible to faults.

A fault is defined to be a defect in some component of the MAS that may lead to a failure when the system is executing. A failure is said to occur when, in a running system, the system produces results that do not meet the requirements of the system [Musa et al. 1987]. Faults can be grouped into five categories as shown in Table 1 [Wagner 2000]. For example, one agent in an MAS encounters a bug that causes it to send a spurious message to a number of other agents. The agents receiving the message do not know how to deal with the message, so they may send further spurious messages to other agents, including the original *buggy* agent. These faults flow through the system eventually resulting in the entire MAS failing.

3 Agent Replication

3.1 What is Agent Replication?

Agent replication is the act of creating duplicates of one or more of the agents in a MAS. Each of these duplicates performs the same task as the original agent. The group of duplicated agents is referred to as a *replicant group* and the individual agents within the replicant group are referred to as *replicants*. There are two types of replication. In *heterogeneous replication* replicants are created which are functionally equivalent but the individual replicants may have been implemented separately. In *homogeneous replica-*

Fault Type	Description
Program bugs	Errors in programming that were not detected by system testing.
Unforeseen states	Omission errors in programming. The programming does not handle a state that can and does occur. System testing did not test for this state.
Processor faults	System crashes or shortages of system resources.
Communications faults	Slow downs, failures or other problems with the communication links.
Emerging behaviour	System behaviour that was not predicted. Emerging behaviour can be beneficial or detrimental.

Table 1: Fault Types

Type	Description	Faults Types
Homogeneous	Agents in a replicant group are identical and deterministic.	Processor, Communications
Homogeneous	Agents in a replicant group are identical and non-deterministic.	Bugs, States, Processor, Communications
Heterogeneous	Agents in a replicant group are not identical, but are designed to perform the same function.	Bugs, State, Processor, Communications

Table 2: Replication Types

tion replicants are exact copies of the original agent. That is, the replicants are functionally equivalent and are copies of the same code.

An agent that uses either homogeneous or heterogeneous replication will introduce a measure of redundancy to the system and increase the fault tolerance. However, homogeneous replication will only increase fault tolerance in certain cases.

If a replicated, deterministic agent encounters a program bug or an unforeseen state, than all of the agents in the replicant group will encounter the bug or unforeseen state. Replication will not have helped in this case. If the agents are non-deterministic, or the percepts received from their environments are different then not all of the replicants will be in the same state, so they may not all encounter the program bug or unforeseen state fault and in this case, replication will increase fault tolerance. If members of the replicant group are run on more than one processor, replication will increase resistance to processor and communications faults. One of the processors may be overloaded or have failed while others have not so those agents can continue running. Communications faults may only be occurring along certain paths so agents on other paths will still be able to communicate.

If heterogeneous replication is used, then fault tolerance to bugs and unforeseen states will be increased. Two agents, programmed separately may not have the same bugs, so a heterogeneous replicant group may be able to continue functioning even if some of the replicants encounter a bug and fail. The case is similar for unforeseen states, one implementation may continue given a certain system state, while another implementation will fail. This property can be exploited when testing new versions of an agent. A replicant group is created that consists of *current* version agents and *new* version agents. If the new agents encounter a bug and fail, the current version agents are still in place and the system will continue to operate. Heterogeneous replication increases fault tolerance to processor and communications faults in the same fashion as homogeneous replication.

3.2 Issues

3.2.1 Inter-Agent Communications and Results Synthesis

Once an agent has been replicated and a replicant group created, communications within the MAS become problematic. If an agent, not part of the replicant group, needs to communicate with the replicant group, it will not know which of the agents to communicate with. If the outside (not part of the replicant group) agent broadcasts messages to all replicants, how will it know how many of the replicants are there? As MASs can be made up of agents from different developers and organizations an agent may not know that the agent it is communicating with is part of a replicant group. How will it handle receiving many replies when it was expecting only one? This raises the issue of results synthesis, that is, the process of taking a number of results and assembling a single result from them.

3.2.2 Read/Write Consistency

Agents always interact with their environments in some way. [Russell & Norvig 1999] When a replicant group exists within a MAS each replicant in the group needs to get percepts from the environment. Each replicant, due to differing processor capacities and loads and varying delays in communication replicants may perceive different data from the environment. This phenomenon is called read consistency [Date 2000]. At one end of the scale, each replicant will get a different value for some piece of data that is needed for its task and each will arrive at different results. At the other end of the scale, for a value that is static in the environment, all replicants will get the same data and hence produce the same results.

Write consistency is closely related to read consistency. When agents are getting percepts from their environments it is expected that they will act upon the environment in some way. Acting on the environment may be considered writing. Since replicants are all doing the same task they will all want to write the same data (maybe even at the same time) to the same place. This can lead to data inconsistency problems. In the classic database example, agent A reads a value, agent B reads the same value, agent a writes a value, now if agent B writes its value it may well be wrong. Or another example, two replicants determine that the action to take is to open valve A while 4 replicants have determined that valve A should remain closed. When action will be taken? Results synthesis can be applied to determine which action should be taken but if the actions are done sequentially then the valve may be left in the wrong state.

A problem closely related to read and write consistency is locking. If agents are reading and writing to their environments locking mechanisms may be in place (for example, the case where the environment is a DBMS) which may result in deadlocks between agents.

3.3 Possible Resolutions

Figure 1 shows the components of an MAS that includes a replicant group.

The sections following give details on how the components may be used to deal with the issues identified in the previous sections. Figure 2 should be referred to while reading the following sections.

3.3.1 Communications Proxy

To address replicant group communications issues a communications proxy may be used. The communication proxy handles all communication between replicants and any other agents in the MAS. To agents outside the replicant group, the replicated agent would appear and act as a single agent. This introduces

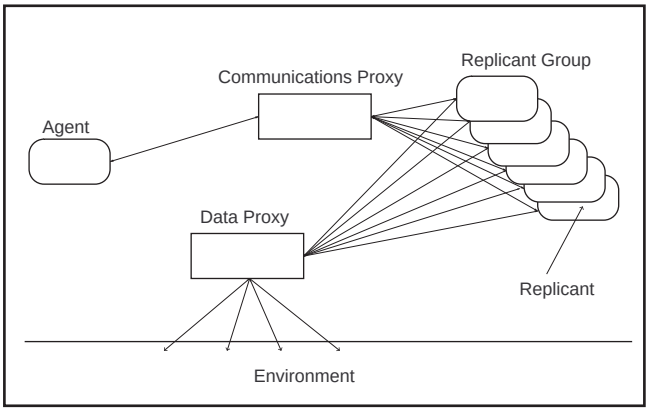

Figure 2: Replicant Group Structures

three new issues. 1. The communication proxy will need to perform results synthesis. If there are N replicants in the replicant group the proxy may get as many as N different results. Logic to determine which result or combination of results to use must be included in the proxy. 2. The proxy will introduce another agent into the system, increasing resource usage. The proxy agent must be able to manage N communications, and the results synthesis. 3. The most serious drawback to using a communication proxy is that it reintroduces a single point of failure. The goal of the replication was to take away the single point of failure through redundancy and the communications proxy would reintroduce it.

One possible way to minimize the single point of failure problem is via adjustable roles. Agents that will be replicated are programmed to be able to perform in three modes: as a single agent, as a member of a replicant group, and as a proxy for a replicant group. If the agent that is currently acting as a proxy fails, the replicants are programmed to detect this and select one of the replicants to adjust its role and take over as the proxy.

3.3.2 Data Proxy

Data proxies closely related to communications proxies and are used to address the problems of read and write consistency. In the case where agents in the MAS are communicating-only agents, data proxies and communications proxies are one and the same.

Data proxies exist between the replicants and the environment. A data proxy can ensure that all replicants receive the same percepts, and can handle results synthesis to ensure write consistency. However, this again introduces new issues. Like a communication proxy, a data proxy is a single point of failure, and a fourth role could be added to the agent to deal with it. Like a communications proxy, a data proxy needs to be able to deal with multiple communications and results synthesis. Data proxies may also need to be able to interact with a variety of environments. For example, reading a sensor, querying a database, or lighting a signal. If the data proxy does one read and distributes that data to each of the replicants the data may not be as up to date as it may have been if the replicants had done an individual read.

3.3.3 State Synchronization

Read and write consistency can be dealt with by keeping all of the replicants in the same state. This state synchronization will insure that all the agents provide the same results, and get the same inputs. This does introduce a lot of complexity and processing overhead to the agents in order to keep the replicants synchronized. This also means that the entire replicant group can only proceed as quickly as the slowest member of the group. If one of the replicants is running on a slow machine or is connected via a slow communications link, this can impact system performance. State synchronization can be achieved at least partially via data and communications proxies. The proxies can control the replicants by careful timing of percepts and messages. State synchronization imposes a degree of structure and central control in the MAS which runs counter to the spirit of MASs. Agents within an MAS are meant to be, at least to a certain degree, autonomous.

3.3.4 Complexity and Resource Usage

All of the solutions identified above introduce complexity and increase system resource usage. The very act of replication increases system resource usage by increasing the number of agents in the system. One way to keep resource usage down is to only allow one or more of the replicants to be active at any one time. If the active replicants are not responding or appear to be acting erroneously, then one of the dormant replicants can be reactivated. This will introduce some problems with state synchronization as once a replicant is revived it needs to be set to the state that the active one was in. If the active one is now failed, it may not be possible to determine the state.

3.3.5 Configurations

Factor	Description	Code
Deterministic	Are the agents in the replicant group deterministic or non-deterministic?	D
Communications Proxy	Does the replicant group employ a proxy agent for communicating with agents outside of the replicant group.	CP
Data Proxy	Does the replicant group employ a proxy for communicating with its environment?	DP
Synchronized	Are the agents in the replicant group kept in the same state?	S
Read Only	Do the agents make any changes to the environment or not?	R
Autonomous	Are the agents autonomous or reactive?	A

Table 3: Factors in Replicant Group Configuration

.

Table 3 lists a set of factors for characterizing a MAS using agent replication. This gives a convenient way to categorize the configuration with a 6-tuple.

4 Experiment

4.1 Description

The purpose of the experiment is to simulate one of the replicant group configurations, to show that replication improves the system failure rate, to gauge the added complexity and resource usage incurred

by replication, and to provide an infrastructure for further experimentation. A simulation was used to allow control over system characteristics such as agent fault rate, processor fault rate and communications fault rate. Using the categorization presented in Table 3, the configuration implemented in the simulation is: $\{-D, CP, -DP, -S, R, -A\}$, that is, deterministic agents, a communications proxy, no data proxy, no synchronization, read only and reactive agents.

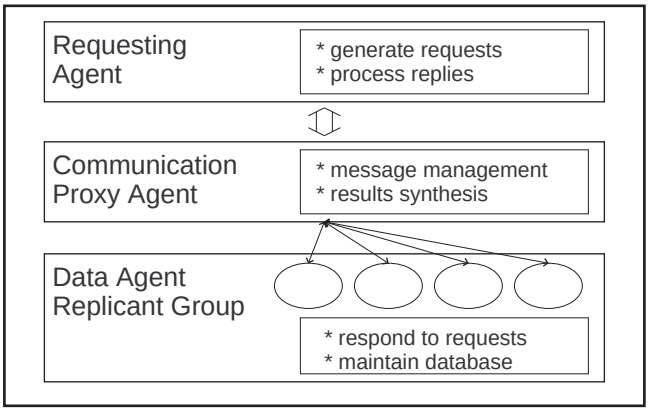

Figure 3: Schematic of Simulation

The MAS implemented is an information query system consisting of two types of agents [Fig 3]. A requesting agent (representing all other agents in the MAS) and a data agent which is able to collect data and respond to queries regarding that data. Agent replication is introduced into the system by replicating the data agent and creating a communications proxy to handle communications and to synthesize results using a voting scheme. The communications proxy passes the requests that it receives to all members of its replicant group. Once a sufficient number of replies are received (in this simulation this is set to 60 percent of the number of replicants) the proxy examines the replies, selects the most popular one and sends it back to the requestor. Ties in the voting are broken by selecting randomly from among the most popular results.

To simulate faults, data agents have a built-in fault rate. The fault rate is expressed as two probabilities, the probability that the data agent has failed, P, and the probability of another type of fault (for example, a bug or unforeseen state), Q. Each time a data agent receives a request it will do one of three things. With probability P it will ignore the request (the agent has failed). With probability Q it will return an erroneous result (the agent has encountered a bug). With probability $1 - (P + Q)$ the agent will return the correct result. For this simulation, P and Q are set to 0.10. To simulate varying load on processors and communications links, a load-simulation factor causes data agents to wait a random length of time before responding to any request.

The experiment consists of running the system for a fixed number of requests, varying the number of replicants in the replicant group. For each run, the mean request response time and the system failure rate is calculated.

The simulation is implemented using DICE to provide the agent infrastructure [Deters 2001]. Communication between agents is done using the DICE post office style messaging facilities and FIPA ACL [FIPA 2000a] style messages.

4.2 Results

The simulation was run with different numbers of replicants, n, in the data agent replicant group, ranging from 1 to 32. Results showing mean request response time and system failure rate versus n are shown in Figure 4.

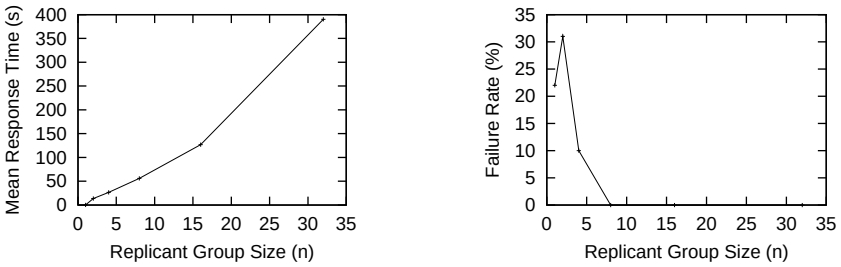

Figure 4: Mean Request Response Time and Failure Rate vs Replicant Group Size

The mean request response time increases linearly with the number of replicants. The system failure rate decreases to 0 once a replicant group size of 8 replicants is reached. The spike in failure rate at $n = 2$ is due to the voting scheme used in the proxy. With only two responses, ties are likely and incorrect results may be returned. The observed system failure rate of 0 for $n \geq 8$ is explained by the independence of each request. Any given request to a data agent has a probability of failure of $P + Q$. A request to the proxy will fail only if more than half of the results considered are incorrect. For $n = 8$, the probability of failure is $R = (P + Q)^3 = 0.008$, and for $n = 16$, $R = 0.000064$. With a trial of 100 requests and $n \geq 8$ it is very unlikely that a failure will appear.

For this configuration, these results show that agent replication improves fault tolerance and lowers the system failure rate at the cost of system overhead and complexity. In this case the number of agents needed to significantly reduce the failure rate is small.

5 Literature

Multi-agent systems are the subject of significant research effort. but little work done directly on agent replication in MASs. Related work can be divided into five areas. 1. Fault tolerance in MASs. 2. MAS structures and organization. 3. Distributed systems and distributed AI research. 4. Software engineering. 5. Agent infrastructures.

Schneider [Schneider 1997] created a method for improving fault tolerance with mobile agents using the TACOMA mobile agent framework. The work was focussed on ensuring that replicated agents were not tainted by faulty processors or malicious entities by the time their results were needed. Kumar et al. [Kumar et al. 2000] present a methodology for using persistent teams to provide redundancy for system critical broker agents within an MAS. The teams are similar to the replicant groups presented here, but, prior to any failures team members are performing individual tasks rather than the same task as in a replicant group. [Hanson & Kephart 1999] presents methods for combatting maelstroms in MASs. Maelstroms are chain reactions of messages that continue without end.

Replication of an agent leads to the creation of some kind of structure within the MAS, so research into MAS structures and organization is closely related to replication. Social organizations look to human organizational models. Examples include societies, co-operatives, coalitions and teams [Ferber 1999][Jennings et al. 1998] The goals of imposing structure and organization into an MAS are usually to improve the performance or capabilities of the system, which is in contrast to the goal of replication

which is to improve fault tolerance. For example, a team may be formed in a MAS, members of the team each have an assigned task and all work to a common goal. Replicant groups look a lot like a team, but each member of a replicant group is assigned the same task.

As MAS are, by nature, distributed, many of the results from distributed systems research can be applied to this research. In particular, distributed database research [Date 2000][Connolly & Begg 1999] can be applied. Certain MASs can be viewed in terms of distributed transaction processing and research such as two phase commit is applicable here. [Jennings & Wooldridge 1999] states that when designing an MAS, one must plan for problems such as synchronization, mutual exclusion for shared resources, deadlock and live lock — these are all issues that distributed systems research has addressed.

The aim of software engineering is to provide structures and tools for handling complexity [Wooldridge 2000]. As argued earlier, MASs are complex, therefore, software engineering techniques need to be applied to MAS development. Several researchers are working on this problem [Jennings & Wooldridge 2001].

There are a number of agent infrastructures available, each with a different focus and different facilities. The FIPA Specification is one of the most commonly used at this time [FIPA 2000b]. DICE was developed to support scalable MASs. DICE uses Java and CORBA to provide a distributed, scalable environment for implementing MASs [Deters 2001]

6 Future Work

This section identifies some possible areas for further investigation.

The simulation presented needs to be expanded and more experiments run. Other configurations of the factors identified in Table 3 need to be implemented. Particular attention needs to be paid to MASs which use autonomous agents.

Replication should be tested using a live system to see if results similar to the one seen in simulation are observed there.

With the overhead that is required to set up and operate a replicant group it is important that replication is only used when it will be effective. Work is needed to create a measure of agent *criticalness*. Only agents with a sufficiently high measure of criticalness would be replicated.

The simulation described above simply loaded a collection of identical agents into the agent infrastructure and set up communications paths as appropriate. What is the appropriate mechanism for starting the replication process? Should an agent be able to replicate itself or should the agent environment handle the replication? How do communications and data proxies get created and how do they learn about the replicants that they represent?

The idea of adjustable roles needs to be further investigated. Agents that are able to perform the four roles identified (single agent, data proxy, communications proxy, replicant) need to be created and tested. Methods of determining when to create a new proxy need to be formulated. Mechanisms for adjusting the roles need to be developed.

Work needs to be done to determine what to do with agents consistently returning poor results. If heterogeneous replication is used it may be possible to kill off agents that are producing poor results and replicate agents that are producing good results.

7 Conclusions

This paper defines agent replication and identifies two main issues that need to be managed (communications and read/write consistency), in a system employing agent replication. Several approaches to resolving those issues are presented, the main one being the use of a proxy agents to act for the entire replicant group in interactions with other agents and the environment.

The technique of agent replication will improve fault tolerance and improve system failure rate at the cost of increased complexity and resource usage. A simulation of the concepts is presented and the simulation shows that replication as a technique shows promise. System failure rates are lowered with replication and very few replicants are needed to lower that rate.

References

[Connolly & Begg 1999] Connolly T. & Begg C. (1999). *Database Systems: A Practical Approach to Design, Implementation, and Management, Second Edition.* Addison-Wesley.

[Date 2000] Date C.J. (2000). *An Introduction to Database Systems, Seventh Edition.* Addison Wesley.

[Deters 2001] Deters, R. (2001). Towards Scalable Multi-agent Systems. University of Saskatchewan, Saskatoon. Draft 2001.

[Ferber 1999] Ferber, J. (1999). *Multi-Agent System, An Introduction to Distributed Artificial Intelligence.* Addison Wesley.

[FIPA 2000a] Foundation for Intelligent Physical Agents. (2000). FIPA ACL Message Structure Specification.

[FIPA 2000b] Foundation for Intelligent Physical Agents (2000). FIPA Agent Management Specification.

[Jennings et al. 1998] Jennings, N., Sycara, K. & Wooldridge, M. (1998). A Roadmap of Agent Research and Development. *International Journal of Autonomous Agents and Multi-Agent Research.* 1(1), pages 7-36. July 1998.

[Jennings & Wooldridge 1999] Jennings, N. & Wooldridge, M., (1999). Software Engineering with Agents: Pitfalls and Pratfalls. IEEE Internet Computing 3 (3) 20-27.

[Jennings & Wooldridge 2001] Jennings, N. R. & Wooldridge, M. (2001). Agent-Oriented Software Engineering. in Handbook of Agent Technology (ed. J. Bradshaw) AAAI/MIT Press. (to appear)

[Kumar et al. 2000] Kumar, S., Cohen, P. R. & Levesque, H. J., (2000). The Adaptive Agent Architecture: Achieving Fault-Tolerance Using Persistent Broker Teams. In *Proceedings, Fourth International Conference on Multi-Agent Systems.* Boston, Massachusetts July, 2000.

[Musa et al. 1987] Musa, J., Iannino, A. & Okumoto K. (1987) *Software Reliability, Measurement, Prediction, Application.* McGraw Hill Book Company.

[Russell & Norvig 1999] Russell S. & Norvig P., (1999). *Artificial Intelligence: A Modern Approach* Prentice-Hall.

[Schneider 1997] Schneider F. B., (1997). Towards a Fault Tolerant and Secure Agentry. *Proceedings, 11th International Workshop on Distributed Algorithms* Saarbucken, Germany, September, 1997.

[Wagner 2000] Wagner, D. N., (2000). Liberal Order for Software Agents? An Economic Analysis. *Journal of Artificial Societies and Social Simulation.* vol 3, no 1, para 1.3, 2000.

[Wooldridge 2000] Wooldridge, M. (2000). On the Sources of Complexity in Agent Design, *Applied Artificial Intelligence.* 14(7):623-644.