Using Dynamic Proxy Agent Replicate Groups to Improve Fault-Tolerance in Multi-Agent Systems

Alan Fedoruk, Ralph Deters
Computer Science Department
University of Saskatchewan
Saskatoon, SK, S7N 5A9 Canada
Phone: 1-306-966-2073
{amf673, deters}@cs.usask.ca

ABSTRACT
Despite the considerable effort spent researching and developing multi-agent systems the actual number of deployed systems is still surprisingly small. One of the reasons for the significant gap between developed and deployed systems is their brittleness. Multi-agent systems are susceptible to all of the same faults as any distributed system, they lack centralized control components, which makes it difficult to detect and treat failures of individual agents, and the agents making up the system are social, thus risking fault-propagation. These faults can impact system performance and lead to overall system failure. Multi-agent systems must be made more dependable before they will be deployed on a large scale. Using redundancy by replication of individual agents within a multi-agent system is one possible approach for improving fault-tolerance, and hence improving reliability and availability—two key components of dependability. Having a group of agents, a replicate group, act like an individual agent leads to increased complexity and system load, and it introduces new challenges to system construction. Using a message proxy, to handle communication for the group, and passive replication strategies effectively deals with the complexity and overhead issues. This paper presents an architecture for implementing agent replicate groups using a message proxy and passive replicate group management. Experimentation and application testing using an implementation of the architecture is presented. The architecture is demonstrated to be a viable technique for increasing dependability in multi-agent systems.

Categories and Subject Descriptors
I.2.11 [Distributed Artificial Intelligence]: Distributed Artificial Intelligence—multi-agent systems.

General Terms
Reliability, Dependability

Keywords
Agent Replication, Dependability, Fault-Tolerance

1. MOTIVATION
Multi-agent systems (MAS) are emerging as an attractive method for building systems in computer science [1]. MASs are highly modular and are well suited to an open, distributed environment—an environment that allows agents created in different organizations to interact over the Internet or other open networks. Although MASs are a promising technology, the number of deployed systems remains small; two reasons for this are the brittleness and performance constraints of MASs [2,4]. A MAS is a complex, distributed system and is prone to the same faults that any distributed system is susceptible to, such as processor faults, communication link failures and slow-downs, and software bugs. However, unlike a traditional distributed system, a MAS is composed of autonomous and social software agents, making fault handling more difficult and making the system more vulnerable to fault propagation. The majority of MAS experimentation is done in a closed and reliable agent environment, which eliminates the need for any fault handling. These environments are well suited for researching system behaviours or new techniques in agent interaction, but they do not represent the open environment in which the MAS will ultimately be deployed. When multi-agent systems are deployed in an open environment—with agents from various organizations interacting in the same MAS and agents distributed over many hosts and communicating over public networks—attention must be paid to fault-tolerance. In an open system, agents may be malicious, poorly designed or error prone, hosts may get overloaded or fail and network connections may be slow or fail altogether. Users expect software to be dependable [3]. To increase dependability—reliability, availability, safety, and security, systems must be able to tolerate a faulty and unpredictable environment: to be reliable, the system must provide a correct result when results are expected; to be available, the system must provide a result in a timely manner; to be secure, the system must be immune to attacks and must not allow unauthorized use; and to be safe, the system must not do anything that will interfere with other systems. Fault-tolerance, though well addressed in distributed systems and other areas of computing, is not well addressed in the MAS literature. This is due to the belief that MASs are fault-tolerant by virtue of their modular and distributed nature, to the fact that most MAS research is focused on investigating system behaviour and new techniques for solving problems, and to the belief that simply having multiple agents provides de-facto redundancy and fault-tolerance. There are two main approaches to fault-tolerance in MASs. Agent-centric (or agent-local) approaches attempt to put fault-tolerance and exception handling into the agents that make up the system. System-centric approaches, on the other hand, build the fault-tolerance apart from the agents. In this approach, an agent, or other framework component, monitors the behaviour of the MAS and takes appropriate steps, should a fault be detected. This method relieves an agent developer of the task of including extensive fault and exception handling into agents. Agent replication, as a fault-
tolerance technique, has aspects of both the agent-centric and system-centric approach. The fault-tolerance is external to the agent; the individual replicate may not even know that it is part of a replicate group. However, since a replicate group is actually acting as a single agent, the fault-tolerance is being provided in an agent-centric fashion. In general, the concept of replication is simple - it provides multiple components in a system, if one fails there are others ready to take over. Agent replication is the act of using more than one agent, each capable of performing the same tasks, as a single agent. Collectively this group of agents is referred to as a replicate group and the individual agents within the replicate group are referred to as replicates. Agents in a group may be different or identical implementations. The number of agents in the group, the replication degree, may change over time. Agents may fail and drop out of the replicate group, others may join to replace failed agents or to boost fault-tolerance at critical times.

2. DYNAMIC PROXY
Agent replication raises three main issues, communication, read/write consistency and overhead. To deal with these three problems the dynamic proxy architecture uses message proxies and passive replicate group management. A message proxy is used to represent a replicate group to the rest of the MAS; the proxy makes the replicate group appear as a single agent. Proxies have three main functions: handle communication between other agents and the replicate group, handle interaction between the agent environment and the replicate group and management of the replicate group. A proxy effectively deals with the communication issue, and some of the read/write consistency issues. However, a proxy adds additional overhead to the replicate group. The number of messages in any interaction will be doubled if passive replication is used and will multiplied N times if active replication is used. Passive replicate group management mitigates two issues. First, having only a single agent active reduces the overhead of the replicate group. Second, having only a single agent active eliminates read/write consistency problems and any problems with replicates interfering with each other. However, passive group management requires state synchronization, which is not always simple and adds overhead to the replicate group. The design goals of the dynamic proxy architecture are: i) to have replicated proxies and group managers so that if the proxy fails, a replicate of it is ready to take over. ii) to create transparent replicate groups, that appear to be a single agent, when in reality a replicate group is present; iii) to manage the replicate groups created via the hot-standby technique; iv) to create replicate groups that consist of variable numbers of agents, possibly distributed over many hosts; v) to create replicate groups that consist of heterogeneous and homogeneous agents; vi) to place as little burden as possible on an agent developer using this architecture. Conceptually, this architecture can be described as follows: a replicate group consists of a number of host agents, each consisting of a message proxy, a group manager and a replicate agent. At any given moment, one of the host agents is the active agent, its proxy is the proxy for the group, and its group manager is responsible for managing the group. Communication with the replicate group is handled by the active message proxy. Host agents hosting new replicates join and leave the replicate group as desired. Replicates in a group may be identical agents or they may be separate implementations, meaning that homogeneous or heterogeneous replication can be performed. The agents may be local to the replicate group, or they may be distributed over more than one host.

3. RESULTS & CONCLUSION
The implementation and experimentation showed that agents can interact with replicate groups in the same manner as they do with non-replicated agents, and that replicated and non-replicated agents can exist and interact in the same MAS. These replication techniques are transparent. The analysis and experimentation showed that the technique is effective at reducing failure rates when agents are in a faulty environment; effectiveness increases with increasing replication degree. In an environment where agents fail with a probability of 0.005, and a replicate group with a degree of N = 4, the replicate group will fail, in the best case, with a probability of 6.25 x 10^-8, that is, a probability of not failing of 0.999999999375. This is the best case and results obtained with an implemented system will be lower, but replication is very effective at even a low degree. The overhead introduced by replicating agents increases with the degree of replication the size of the object representing the replicated agent state, and the level of faultiness of the agents and their environments. The analysis of replication in the controlled experiments showed that the overhead was significant - more than 100%. However, the application testing showed by selecting good migration strategies of replicates the interaction-overhead can be reduced to reasonable levels (14%).

How much overhead is acceptable is of course application dependent. In some cases, no extra overhead can be tolerated, but in others the fault-tolerance gained outweighs even large amounts of additional overhead.

Tools to implement agents using dynamic proxy architecture are provided. Implementing an agent requires a developer to sub-class an abstract agent class and to provide a small number of methods (set and get state). Replicating agents, using the dynamic proxy architecture, is a viable technique. Replication imposes a measure of overhead as discussed above; however, it can also be used to manage performance in certain circumstances. This can be accomplished by managing the location of the active agent. If the active agent is closer to the majority of agents it is interacting with, response times can be improved.

4. REFERENCES