

Group Membership Services for Dynamically Organized Sensible Agent-Based Systems

K. S. Barber R. M. McKay T. H. Liu

The Laboratory for Intelligent Processes and Systems
Department of Electrical and Computer Engineering, ENS 240
University of Texas at Austin
Austin, TX 78712-1084
barber@lips.utexas.edu

Abstract

To meet the requirements of dynamic and uncertain environments, Sensible Agents are capable of dynamically changing their organizational structures by modifying their roles, which are represented as their autonomy levels (e.g. command-driven, consensus, locally autonomous, and master). Maintaining correct knowledge of the current group membership is essential for Sensible Agents since such information impacts how an agent makes its decisions. This paper presents a group membership service designed for Sensible Agent-based systems to maintain agents' model of group membership and to tolerate communication failures, process failures, and multiple partitions. A particular agent is selected as the manager to coordinate every agent's view of the group membership. An algorithm (including merging partitions, suspecting and proving of agent failures) is presented to maintain group membership when agents join, leave, or partitions happen. A discussion of current work to implement our group membership services in a distributed simulation environment for Naval Radar Frequency Management domain is also discussed.

Introduction

The Sensible Agent model is a multi-agent system designed for dynamic and uncertain environments, which require the system to be flexible and adaptive to situation and environment changes (Barber, 1996). Sensible Agents achieve these qualities by representing and manipulating the structure of the interaction frameworks in which they plan to achieve their goals. Modeling the behaviors and intentions of other agents is essential for Sensible Agents. Two related research issues that affect how agents make decisions are the organizational structure design and group membership maintenance. The former issue concerns how the agents are organized for specific tasks (e.g. hierarchical, peer group, etc.) (Glaser and Morignot, 1997). The later is concerned with the maintenance of a group boundary when agents can join, leave, or even crash dynamically (Jahanian et al., 1993).

This paper focuses on group membership maintenance. Here, the notion of a group is abstracted to include any kind of organizational structure. The group membership service provides only the information needed to identify whether an agent is within the organization or not. Possible reasons for forming groups include specialization (agents with different abilities grouped to achieve a shared goal), fault tolerance (agents with duplicated capabilities teamed for a specific goal), reduction of communication, or reduction of system complexity. For reasons like the crash of an agent process or a communication link failure, agents may fail to fulfill their duties to their group. When this happens, it is important for the rest of the group to know and to respond appropriately to avoid endlessly waiting for messages that will never come. On the other hand, if the suspected agent is temporarily inaccessible due to a re-routable communication link failure, the group may want to establish another communication path to re-connect with that agent.

Our example domain is Naval Radar Frequency Management (NRFM). In this domain, there are several geographically distributed ships, each with an associated radar. The available frequency range for these radars is limited. Each ship has a Sensible Agent controlling its radar. As the ships move from one point to another, they must attempt to minimize the interference affecting their own radar, while also trying to minimize the amount of frequency interference they produce for other agents. Sensible Agents in this system will form various groups to work on this problem. For example, they might form a peer group in which all the members negotiate for their desired radar frequency. On the other hand, several agents might decide to appeal to a single, more capable agent, to create a frequency assignment for the whole group. Regardless of the type of group formed, membership can change dynamically due to a variety of factors, such as ships becoming disabled or sinking, loss of communication, new agents joining, old agents deciding to leave, etc. Because of these possibilities, current, accurate group membership information is very valuable.

Researchers in distributed computing have tackled the issues involved with ensuring that all the members in a group have a consistent view of the Group Membership

(GM). Various approaches with different features have been proposed. The main idea is that processes multicast messages to group members to track membership changes (e.g. new processes join, existing processes leave). However, certain issues need to be addressed before applying existing GM techniques to Sensible Agent systems. These include the types of failures tolerated, the mechanism used to reach agreement on group membership changes, and security. To address these issues, we propose a partitionable group membership algorithm for Sensible Agents based on existing research results. A brief background review is provided in the following section. Solution requirements are described in Section 3. Section 4 presents our approach and algorithms. The effects of GM services on Sensible Agent behavior are discussed in Section 5. Section 6 introduces our implementation environment. Finally, Section 7 concludes the paper and discusses future work.

Related Research

Sensible Agents

Many domains challenge agent-based systems to be adaptive and flexible while providing consistently efficient and effective problem solutions. The Sensible Agent model has been proposed to answer the demands of these domains (Barber, 1996). Sensible Agents are capable of Dynamic Adaptive Autonomy, meaning they have the ability to form groups, called Autonomy Groups, to work on joint goals. Each goal an agent is working on is assigned an Autonomy Level, which specifies the Autonomy Group structure. Several typical structures are named: Master/Command-driven – hierarchical, Consensus – peer group, and Locally Autonomous – agent works alone. By forming these groups, Sensible Agents modify the overall problem-solving organization of their system. These modifications serve to increase system flexibility and maintain or improve solution quality.

As shown in Figure 1, Each Sensible Agent is composed of four major modules (Barber, 1996). The Action Planner (AP) module solves domain problems and executes problem solutions. The Autonomy Reasoner (AR) determines the appropriate autonomy level for each goal. It is also responsible for establishing the goal's corresponding Autonomy Group. The Conflict Resolution Advisor (CRA) identifies, classifies, and generates possible solutions for conflicts occurring between the self-agent and other agents. The Perspective Modeler (PM) contains the agent's explicit model of its local (subjective) viewpoint of the world. The overall model includes the behavioral, declarative, and intentional models of the self-agent (the agent who's perspective is being used), other agents, and the environment. Other modules within the self-agent can access the PM for the current status of its models.

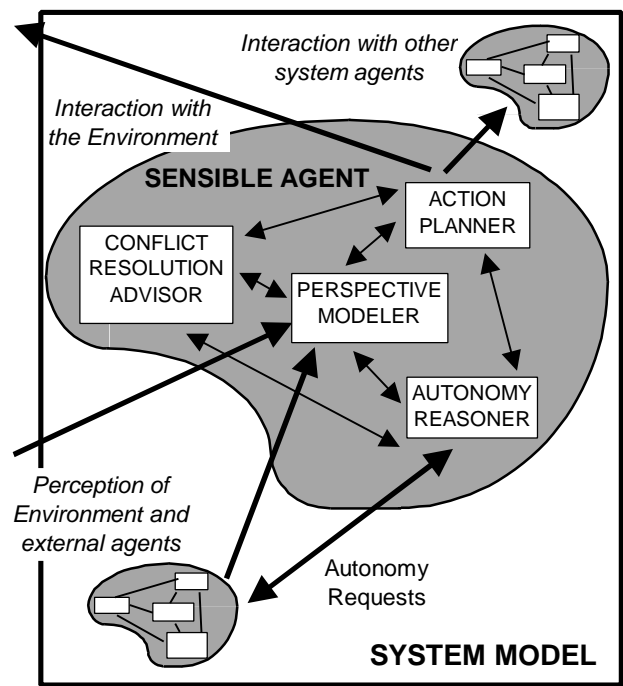


Figure 1 The Architecture of Sensible Agent

Group Membership in Distributed Computing

In the Distributed Systems research community, a large amount of work has developed group membership and communication services for distributed computer processors and processes, such as Isis (Birman and van Renesse, 1994), Horus (van Renesse et al., 1996), Transis (Dolev and Malki, 1996), Totem (Moser et al., 1996), Newtop (Ezhilchelvan et al., 1995), and Relacs (Babaoglu et al., 1995), which offer different guarantees about the reliability and order of message delivery. The main idea is that processes multicast messages to group members to track membership changes (e.g. new processes join, existing processes leave).

A group expels members that have failed. There are many types of failures a process can be exposed to. Perfect accuracy of failure detection cannot be guaranteed due to the fact that there is no message delay limitation (in time) for asynchronous distributed systems and it is impossible to distinguish slowness from a crash. The concept of a failure suspector has been devised to encompass all of the impossibility (Babaoglu et al., 1997). Every process has a failure suspector, which is responsible for informing its associated process of the suspected failures of other group members. The failure suspector may have some wrong suspicions, resulting in the expulsion of a live member.

Different design assumptions about partitions have been proposed. The primary partition model assumes there is only one main partition, or set of processes, that can communicate and work together. Agents outside the primary partition are considered crashed and are ignored.

The group membership protocols developed in (Ricciardi and Birman, 1995) make this assumption. The multiple partition model (e.g. (Babaoglu et al., 1997)) allows the initial group to be instantly divided into several subgroups that cannot communicate with each other. These subgroups are still expected to make progress on their own, and will merge if they discover a way to communicate.

Requirements and Assumptions

This section discusses our requirements for a group membership algorithm for Sensible Agents. First, the algorithm has to tolerate multiple partitions. Since it could be the case that none of these subgroups has a majority of the original group, the typical mechanism used in a primary partition model to reach agreement, voting, cannot be used here. Any subgroup without a majority of members would deadlock, unable to decide on a course of action.

The second set of requirements is the types of failures our algorithm should tolerate. We consider two points of failure – processes and communication links. By process, we mean a Sensible Agent, and by link, we mean a one-way communication channel between two processes that can fail independently of the processes. For both of those, we use the most common failure model in the distributed computing community, the crash failure. By crash failure, we mean a complete and permanent stoppage of all

functionality, with no warning. Notably missing from our failure model is the Byzantine failure, or arbitrary incorrect behavior. This failure type is out of the scope of this paper.

Third, the cost of expelling a live agent from the group is high. Therefore, a mechanism is required to give suspected agents a chance to prove that they are still alive. This will help decrease the number of agents wrongly expelled. However, we should not diminish our algorithm's ability to remove members that actually have failed or are no longer reachable.

Finally, we assume group members have a secure method for verifying the identity of other agents. Our proposed implementation of this assumption is a security service layer that provides message encryption services (e.g. public-key, secret-key) for our group membership service (Schneier, 1996).

Approach

In the proposed group membership algorithm, every group contains a distinguished agent called a manager (Ricciardi and Birman, 1995). This agent is responsible both for coordinating the agents' views of the membership and for being the group's contact with agents outside the group. The use of this agent, as opposed to having all group members broadcast all messages, reduces the communication costs of our algorithm from $O(n^2)$ to $O(n)$.

Incoming Event	Response	Notes
from Manager: receive submit(add(<i>newmember</i>))	Pre-commit, send back to Manager ack(submit(add (<i>newmember</i>)))	
from Manager: receive commit(add(<i>newmember</i>))	Commit to new view that includes <i>newmember</i>	Should always come after submit(add <i>newmember</i>)
from Manager: receive submit(remove(<i>currentmember</i>))	Attempt to forward the message to <i>currentmember</i>	
from Manager: receive commit(remove(<i>currentmember</i>))	Commit to new view that excludes <i>currentmember</i>	If received with no accompanying submit, <i>currentmember</i> voluntarily left the group
from <i>currentmember</i> : receive object(submit(remove(<i>currentmember</i>)))	Forward to Manager	
from Manager: receive retract(submit(remove(<i>currentmember</i>)))	Reset suspicions of <i>currentmember</i>	
From Manager: Receive commit(disband(<i>newmanager</i>))	1. Form self-group, of which this agent is the manager 2. Attempt to merge with the group having <i>newmanager</i> as its manager	
From Failure Suspector: Suspect(1, <i>currentmember</i>)	Send submit(suspect(<i>currentmember</i>)) to <i>currentmember</i> , or Send submit(suspect(<i>currentmember</i>)) to Manager	The 1 in the suspect event indicates that this agent has not yet attempted to verify its suspicion with the suspected party
from Failure Suspector: suspect(2, <i>currentmember</i>)	Send submit(suspect(<i>currentmember</i>)) to Manager	The 2 in the suspect event indicates that this agent has already attempted to verify its suspicion with the suspected party
from Failure Suspector: suspect(2, <i>currentmanager</i>)	1. Form self-group, of which this agent is the manager 2. Continue trying to make progress by itself until it encounters another partition, then send request(add(<i>thisagent</i>)) to the manager of that partition.	Special case of the above procedure when <i>currentmember</i> is <i>currentmanager</i>
from <i>nonmember</i> : receive request(*)	Forward to Manager	* is any type of group membership change, such as add, merge, etc.

Table 1: Group Membership Algorithm for Non-Manager

Incoming Event	Response	Notes
from <i>currentmemberA</i> : submit(suspect(<i>currentmemberB</i>))	1. Broadcast submit(remove(<i>currentmemberB</i>)) to all members 2. Wait until either a. Receive object(submit(remove(<i>currentmemberB</i>))) b. Receive ack(submit(remove(<i>currentmemberB</i>))) from all live members and some predefined time <i>t</i> has passed	If manager has reason to believe that <i>currentmemberB</i> is still alive, it may simply inform <i>currentmemberA</i> of this fact. It may propose a new communication route to <i>currentmemberA</i> .
from <i>currentmemberB</i> : object(submit(remove(<i>currentmemberB</i>)))	Reset suspicions of <i>currentmember</i> , broadcast Retract(submit(remove(<i>currentmemberB</i>))) to all members	
Receive ack(submit(remove(<i>currentmemberB</i>))) from all live members and some predefined time <i>t</i> has passed	Broadcast commit(remove(<i>currentmemberB</i>)) to all members	
from <i>currentmember</i> : request(remove(<i>currentmember</i>))	Broadcast commit(remove(<i>currentmember</i>)) to all members	
from <i>nonmembermanager</i> : receive request(merge(<i>nonmembermanager</i>))	Start merge procedure (see paragraph below)	Decision of whether to allow a new agent into the group is made by the application layer.
from Failure Suspector: suspect(1, <i>currentmember</i>)	Send submit(suspect(<i>currentmember</i>)) to <i>currentmember</i>	The 1 in the suspect event indicates that this agent has not yet attempted to verify its suspicion with the suspected party
from Failure Suspector: suspect(2, <i>currentmember</i>)	Same as if manager received submit(suspect(<i>currentmemberB</i>)) from <i>currentmemberA</i>	The 2 in the suspect event indicates that this agent has already attempted to verify its suspicion with the suspected party

Table 2: Group Membership Algorithm for Manager

Agents report their suspicions of other agents to their manager and if there is no evidence to object to that suspicion, the manager will broadcast that suspicion. When agents receive such a broadcast, they will send an acknowledgement back to their manager (otherwise they will be suspected as well) and attempt to forward this message to the suspected agent, using the available communication links. To address our first and third requirements, we have devised a mechanism using an objection and proof methodology. Using this mechanism, any agent receiving a message indicating the group's suspicion of it has the option of objecting to the manager. When the manager receives such an objection, it will broadcast a retraction. Our use of this mechanism helps motivate our second set of requirements, because we then allow the group to assist those agents experiencing communication difficulties (slow or failed link) by searching for new communication links.

When an agent loses its connection with its manager (either the manager crashes, communication link brakes, or partitions happen), it will form a temporary single agent group of which it is the manager. Then this manager tries to contact other agents/partitions, with the objective of merging with them. After an unstable time period, all the reachable partitions will have merged. By this method, we can guarantee there is always a manager and therefore the group membership service is always available. The following tables describe our algorithms.

When two partitions become aware of each other, they may execute a merge. The first step in a merge is to decide which current manager will be the manager of the new group. This can be decided in a variety of ways, such as a manager rank, number of members in each partition, etc.

Once this is decided, the members of the partition that does not contain the new manager are told by their current manager to disband and join the new partition individually. This avoids the problem of getting all the members to agree on joining the new partition.

Effects of Group Membership on Sensible Agents

The provision of a GM service affects the way a Sensible Agent behaves. This change in behavior can be traced to each of the modules in an agent. The GM service is located inside the Perspective Modeler, which provides the information to the other agents about the current membership for all the groups of which the self-agent is a member.

For any multi-agent Autonomy Group, current information on group membership is vital to the Action Planner's function. Its plan may be completely different depending on the agents in the group. For example, consider a Consensus Group that has already decided on a group plan for its joint goals. If a member dies after having been assigned some tasks, the entire group plan may need to be re-thought to account for this loss.

The Autonomy Reasoner may decide that the agent should leave an Autonomy Group (typically for some penalty). One good reason for leaving an Autonomy Group is if vital members have failed. For example, if an agent is command-driven for a goal and its master fails, it should not simply remain idle waiting for instruction. It should reevaluate its desired Autonomy Level, perhaps entering into a new agreement with other agents. This type of

flexibility is facilitated by current information about who is still alive in a group.

The kinds of conflict resolution strategies offered by the Conflict Resolution Advisor vary widely depending on the types of Autonomy Groups the agent is a member of. For example, if it is in a Consensus Group, a conflict with another agent may require negotiation. However, a conflict with a fellow command-driven agent may best be resolved by appealing to the superior intellect of a master agent. In addition to the types of groups of which an agent is a member, the actual membership of those groups may be a factor in choosing a conflict resolution strategy. For example, a fellow Consensus Group member may be skilled at mediating conflicts, so the CRA might suggest taking advantage of that.

Implementation

This section introduces the Sensible Agent Testbed, which is the Sensible Agent implementation integrated with a distributed simulation environment (Figure 2). We use the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) standards and the OMG Interface Definition Language (IDL) to formally define the interactions among agents and their modules. The use of IDL permits continued Sensible Agent evolution in an implementation-language-independent manner and facilitates parallel research initiatives within the framework of an integrated distributed object system. Sensible Agent modules and the environment simulator are implemented as CORBA objects that currently communicate through the Xerox Inter-Language Unification (ILU)(Xerox, 1998) distributed object environment (i.e. Object Request Broker, ORB). The Sensible Agent System Interface (SASI) provides a functional wrapper around the internal modules. This implementation allows the integration of multiple implementation languages (C++, Java, Lisp and ModSIM) simultaneously on Solaris, WindowsNT and Linux platforms. Additionally, ORB support of the CORBA Internet Inter-Orb Protocol (IIOP) standard will allow the connection of external ORBs to the Sensible Agent Testbed, further enhancing its extensibility. The use of IIOP combined with the OMG's CORBAServices Object Naming Services (COSNaming) allows the presentation of a public interface to the Sensible Agent Testbed, permitting external entities to connect with the testbed for experimentation.

In the simulation Naval Radar Frequency Management, a group of ships moves from a point out in the ocean to a point on the coast. Along the way, they form and dissolve various groups to work on their goal of minimizing radar interference and their other joint goals. Sometimes the dissolution of a group is due to the fact that the goals have been accomplished. However, storm may disrupt the agent's communication. In this situation, the agents realize that the system has become partitioned, so they revert to their self-groups in order to avoid deadlock and continue

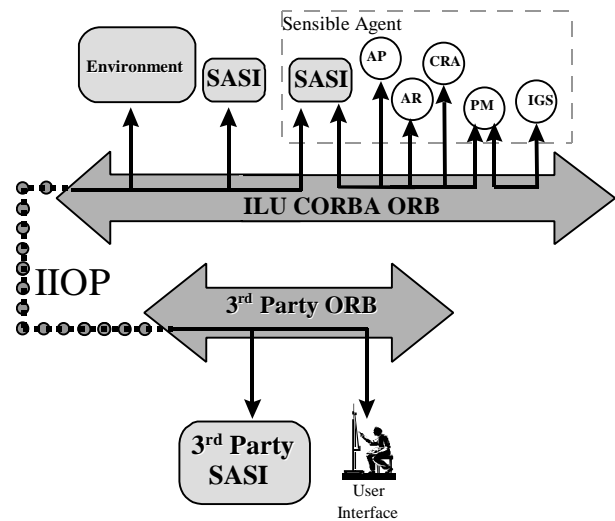


Figure 2: Sensible Agent Testbed Implementation Architecture

making progress. When the ships can communicate again, they have the option of merging their partitions. Currently we are working to fully implement our group membership service. Further analysis of the performance data collected will help to validate our algorithm.

Conclusion

This paper presented a group membership service for a Sensible Agent-based system to maintain correct and consistent models of group boundaries. Based on the existing group membership techniques, our algorithm uses an objection and proof mechanism to improve safety (live agents will not be kicked out) without sacrificing liveness (the failed agent will eventually be kicked out).

The group membership service helps Sensible Agents carry out their goals. Since a Sensible Agent's planning and execution behaviors are highly dependent on the interaction among agents, the correct maintenance of the membership helps agents to react correctly to organizational changes. These changes might include: new agents joining a group (to pursue a shared goal), existing agents leaving the group (either due to being expelled or simply deciding to leave), and the formation of multiple partitions (a group is partitioned into several sub-groups). Sensible Agents already possess the ability to form new Autonomy Groups in response to these changes, however, they still require a service that provides notification of changes in the group's composition – our group membership service.

Future work in this area will extend to the maintenance of more complex organizational structures, including groups within groups (abstractional hierarchy) and overlapping groups. One of the main reasons for forming hierarchical groups is to enable command and information abstraction. We would also like to expand our failure model consideration to include Byzantine failures, or arbitrary

behavior. In addition, simulation and additional experimental data will be further analyzed regarding the impact on performance of the autonomy level changes as well as organization changes. The information gained during execution regarding perceived performance of the agent can be used to help Sensible Agents make better decisions about autonomy levels.

Acknowledgements

The work presented here is sponsored in part by the Texas Higher Education Coordinating Board project #003658-415

Reference

Babaoglu, O., Davoli, R., Giachini, L., and Baker, M. 1995. Relacs: a Communication Infrastructure for Constructing Reliable Application in Large-Scale Distributed Systems. In Proceedings of the Proc. of Hawaii International Conference on Computer and System Science, 612-621. :

Babaoglu, O., Davoli, R., and Montresor, A. 1997. Partitionable Group Membership: Specification and Algorithms, , UBLCS-97-1, Department of computer science, University of Bologna.

Barber, K. S. 1996. The Architecture for Sensible Agents. In Proceedings of the International Multidisciplinary Conference, Intelligent Systems: A Semiotic Perspective, 49-54. Gaithersburg, MD: National Institute of Standards and Technology.

Birman, K. P. and van Renesse, R. 1994. Reliable Distributed Computing with the Isis Toolkit. Los Alamitos, CA: IEEE Computer Society Press.

Dolev, D. and Malki, D. 1996. The Transis Approach to High Availability Cluster Communication. Communications of ACM 39(4): 64-70.

Ezhilchelvan, P. D., Macedo, R., and Shrivastava, S. 1995. Newtop: a Fault-Tolerant Group Communication Protocol. In Proceedings of the Proc. of IEEE International Conference on Distributed Computing Systems, 296-306. :

Glaser, N. and Morignot, P. 1997. The Reorganization of Societies of Autonomous Agents. In Multi-Agents Rationality: Proceedings of the Eighth European Workshop on Modeling Autonomous Agents in a Multi-Agents World, Boman, M. and van de Velde, W., Eds. New York: Springer-Verlag, 98-111.

Jahanian, F., Fakhouri, S., and Rajkumar, R. 1993. Processor Group Membership Protocols: Specification, Design and Implementation. In Proceedings of the Proceedings of the 12th Symposium on Reliable Distributed System. :

Moser, L. E., Melliar-Smith, P. M., Agarawal, D. A., Budhia, R. K., and Lingley-Papadopolous, C. A. 1996. Totem: a Fault-Tolerant Multicast Group Communication System. Comm. of the ACM 39(4): 54-63.

Ricciardi, A. M. and Birman, K. P. 1995. Process

Membership in Asynchronous Environments, , TR93-1328, Cornell University.

Schneier, B. 1996. Applied Cryptography: Protocols, Algorithms, and Source code in C, 2nd edition ed: John Wiley & Sons.

van Renesse, R., Birman, K. P., and Maffeis, S. 1996. Horus: a Flexible Group Communication Services. Comm. of the ACM 39(4): 76-83.