Fault-Tolerant Group Membership Protocols using Physical Robot Messengers

Rami Yared[1], Xavier Défago[1,2], and Takuya Katayama[1]

[1]School of Information Science, Japan Advanced Institute of Science and Technology (JAIST)
[2]PRESTO, Japan Science and Technology Agency (JST)

December 1, 2004
IS-RR-2004-019

Fault-tolerant group membership protocols using physical robot messengers

Rami Yared[*], Xavier Défago[*,†], and Takuya Katayama[*]

[*]School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Tatsunokuchi, Ishikawa 923-1292, Japan

[†]PRESTO, Japan Science and Technology Agency (JST)

Email: {r-yared,defago,katayama}@jaist.ac.jp

Abstract

In this paper, we study the group membership and view synchrony problem in a distributed system composed of a group of teams of mobile robots communicating by physical robot messengers.

Communication by robot messengers raises new issues and relevant fault tolerance techniques that are different from those in traditional distributed systems.

1 Introduction

In this paper, we define a distributed system composed of a group of teams of cooperative autonomous mobile robots, communications between teams take place by sending physically a robot messenger from the team sender to the team receiver. A distributed system composed of teams of autonomous mobile robots communicating by robot messengers has distinct aspects and issues that are different of those in conventional distributed systems.

comparison with traditional distributed systems : a team of mobile robots maps to a process in traditional distributed system, with a difference that a team can not send any message, if its pool of messengers is empty, unless it receives a messenger from another team, while a process can send messages at any time.

Obviously, communications by messengers take longer delays and larger transmission time between nodes, compared to conventional communication media such as radio, electric signals, and infrared beams.

A remarkable advantage of teams of mobile robots, is that a robot messenger has enough memory to carry any quantity of available of messages from a team source to a team destination, in contrast to bandwidth limitations for communication channels in traditional distributed systems.

Concerning fault-tolerance aspects, we assume in this paper that either teams or messengers fail by crash. a team failure maps to a process failure, and a messenger failure corresponds to lossy channels, with the major difference resides in fault-tolerant techniques. In order to tolerate a bounded number of faulty messengers, the team sender sends a set of messengers such that its cardinality is greater than the maximal number of faulty messengers in the system, so this method guarantees a reliable communication channel.

On the other hand, there is a particular failure detection technique that is relevant to distributed system composed of teams of mobile robots communicating by messengers, that is different from conventional failure detection mechanisms. It is well-known that it is impossible to correctly and deterministically detect a process crash in purely asynchronous distributed systems [5], because it is impossible to distinguish between a crashed process and a very slow one. But in a context of robots communicating by physical messengers, the detection of a crashed team occurred locally on its site, by at least one correct messenger, and consequently the system is augmented with some perfect failure detector.

Example Let us illustrate the motivations of this approach with a simple example. Consider a distributed application composed of a group of teams of cooperative mobile robots searching mineral objects inside a mine. In this underground application there is no established radio communications infrastructure, also it is not practical at all to establish any radio communication system like (e.g., [2]) in this environment. [1] Using ultrasonic sound media in this situation is not feasible. On the other hand communicating by infrared technology (e.g., [7]) can solve the problem of the absence of radio communication infrastructure, but it needs a line-of-sight between communicating robots, and signals could be interrupted because of moving obstacles.

So, it is convenient to communicate by physical robot messengers in such applications, and also in similar environments, like underwater and spatial applications. Also, communications by messengers could be used to tolerate catastrophic crashes of a whole radio or infrared communicating system between teams of robots.

Group Membership and View Synchrony In a distributed system composed of teams of mobile robots, with presence of failures, it is desirable to establish a group membership and view synchrony protocol to:

- Allow teams to join and/or leave a group in a consistent manner.
- Enable teams to install a new view such that all teams in the system agree on every new installed view.

[1]the communication infrastructure in [2] is developed for autonomous vehicles applications.

So, a group membership and view synchrony protocol must generate an ordered sequence of consistent views.

Definitions We define a group as a set of teams which are said to be members of the group. A team becomes a group member by requesting to *join* the group; it can cease being a member by requesting to *leave* the group. A view is the output of membership service, consisting of the list of the current members in the group, and a sequence number.

Contribution In this paper, we define a distributed system composed of teams of cooperative autonomous mobile robots, such that the inter-team communications “communications between teams” occur by sending robot messengers.

We assume that the system is completely connected, so there exists a “communication route” between each pair of teams, and every team has a pool of robot messengers for sending messages to other teams. In our system model, a group is a set of teams communicating by robot messengers, we present and discuss a distributed algorithm which is the group membership and view synchrony in this system model prone to messenger failures and team failures.

We handle the *join* of a new team to a group, the *leave* of a team, the establishment and confirmation of a new view in the system. We discuss three models of failures, the first model assumes that messengers and teams are both correct, the second model handles the failures of messengers assuming that all the teams are correct, and the third is the most general model in which we consider both messengers and teams failures.

We present a group membership algorithm and give arguments showing its correctness, then we evaluate briefly the required energy and time, to run the algorithm in each failure model.

Related work There exists group membership and view synchrony protocols for conventional distributed systems. Schemmer et al. [9] developed an architecture allowing mobile systems to schedule shared resource in real-time based on wireless communications, they present two membership protocols which allow mobile systems to *join* and *leave* a group with predictable delay at any time, these protocols dynamically allocate bandwidth to joining stations. Their approach aimed at solving the problem of congestion in traffic control systems. Briefly, many protocols has been presented (e.g., [1,4,8,9]) based on different scheduling techniques to allocate shared resources such as the bandwidth to joining stations for dynamically changing groups.

Abstractions of the existing protocols concerning group membership and view synchrony for traditional distributed systems, cannot be adapted to our distributed system model. On the other hand, the mobile agents approach is based on software migration, consequently this agent could be easily replicated, but in our model a messenger is a physical entity. Mobile agents approach does not meet our system model and requirements.

Structure The rest of the paper is organized as follows. Section 2 describes the system model and the basic concepts and assumptions. Section 3 describes the three failure models, the failure-free, messengers failure, and both teams/messengers failure models. Section 4 describes our group membership algorithm with correctness arguments and behavior evaluation for each failure model, and Section 5 concludes the paper.

2 System model & definitions

2.1 System model

We consider a distributed system composed of a group of n teams of autonomous mobile robots and m messengers. n and m are > 1. These teams cooperate with each other to achieve a required task determined by the upper layer. The system is purely asynchronous, so there exists no bounds neither on the speeds of processing information by teams, nor on the messages delays. Teams communicate between each other by exchanging robot messengers. Figure 1 illustrates our system model.

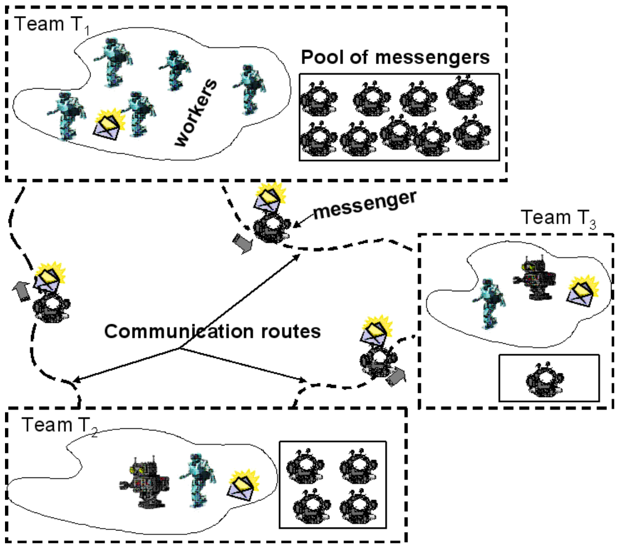

Figure 1: System model

We assume that there exists a "communication route" between each pair of teams, such that each team can communicate directly with other teams, and the system is completely connected.

The system (S) is a group of teams $S = \{T_1, T_2, \ldots, T_n\}$. Every team has an identifier, a set of robots named "workers" responsible for executing the required tasks, and a pool of robot messengers.

In this model, a robot messenger is ready to transmit messages on behalf of its team and also on behalf of other teams, and we assume that the capacity of memory of a messenger is large enough such that it can carry all available messages.

When a team receives a message, the cardinality of its pool is incremented by one, and it is decremented by one when it sends a message. We assume that each messenger has enough energy to move two hops at most, after that it requires a power supply from any team in the system.

2.2 Metrics

In addition to the complexity metrics used in traditional distributed systems, we consider a new metric that we call *energy complexity*.

Roughly speaking, the energy complexity of an algorithm A measures the total amount of energy spent by a single run of the algorithm. The energy is evaluated by counting the number of hops [2] that must be made by messengers until the algorithm terminates.

2.3 Group membership & view synchrony

The membership service maintains a list of currently active a connected processes, in failure-prone distributed systems, and delivers this information to the application whenever it changes. The reliable multi-cast services deliver messages to the current view members. For more information on the subject, we refer to the survey of Chockler et al. [3]. A group membership can also be seen as a high-level failure detection mechanism that provides consistent information about suspicions and failures [6, 10]. In short, a group membership keeps a track of what a process belongs to the distributed computation and what process does not.

In a distributed system composed of teams of robots communicating by messengers, a group membership service provides a list of non-crashed teams that currently belong to the system, and satisfies three properties: validity, agreement and termination. Validity is explained as follows: let v_i and v_{i+1} be two consecutive views, if a team $T_i \in v_i \setminus v_{i+1}$ then some team has executed *leave(T_i)* and if a team $T_i \in v_{i+1} \setminus v_i$ then some team has executed *join(T_i)*. The agreement property ensures that the same view would be installed by all the teams of the group (agreement on the view) since agreement on uniquely identified views is necessary for synchronizing communications. The termination means that if team T_i executes join(T_q), then unless T_i crashes, eventually a view v' is installed such that either $T_q \in v'$ or $T_p \notin v'$. We present the following notations used in the paper:

- $|T_i|$ is the number of messengers exist in the pool of the team T_i.
- initiator is the team which proposes (join) or a (leave) operation, and consequently initiates a procedure of creating a new view.
- logical ring is a logical circular list of teams identifiers.
- v_{ini} is the initial view of the system.
- v_{act} is the current view of the system.
- v_{fin} is the resulting view of the system.

[2]We call "*hop*" the journey from one team to another made by some messenger.

3 Failure Models

We consider that a messenger and a team fail by "crash", and discuss three possible models of failure. In the Model A, we assume that all messengers and teams are correct. In Model B we consider the crash of messengers only, but all the teams are correct. We present the most general case by the Model C, which considers both teams and messengers crashes.

3.1 Model A: Failure-free

Model A considers the case of failure free, in this model there are neither crash of messengers, nor crash of teams. Model A is specified by the the two following properties:

- property A1: All messengers are correct.
- property A2: All teams are correct.

3.2 Model B: Messenger failure

In this model, we consider the failures of messengers only, so a robot messenger may fail by crash while it moves between teams carrying messages. (no crash of teams in Model B).

We assume that number of faulty messengers is bounded, and denote this upper bound by $\overline{M}$.

In this model, We have the following properties:

- *property B1*: A messenger can fail by crash, and when it crashes, this crash is permanent.
- *property B2*: A whole team of robots never crashes, so all the teams in the system are correct.

- *property B3*: There is at least one correct messenger in the system, so ($\overline{M} < m$).

3.3 Model C: Team/messenger failure

In this model, the system contains *some* faulty teams and *some* faulty messengers. We assume that the number of faulty teams and faulty messengers is bounded, we denote the maximal number of faulty (teams, messengers) in the system by: ($\overline{T}, \overline{M}$).

In this model we have the following properties:

- *Property C1*: A whole team(s) may fail by crash and when a team crashes, this crash is permanent.

- *Property C2*: A correct messenger never crashes while doing its jobs by carrying messages between teams, but if its team has crashed and it was *inside* it at the moment of crash, then the correct messenger crashes with its team. [3] Correct messengers that were *outside* their teams never crash.

- *Property C3*: The crash of a team implies the crash of all the *robots* inside that team, the crashed robots can be classified in two categories: the workers of the crashed team, and its messengers *either faulty or correct* which are still *inside* the team at the moment of crash.

- *Property C4*: There is at least one correct team, and at least one correct robot messenger in the system, we can express this condition as follows: $\overline{T} < n$ and $\overline{M} < m$.

- *Property C5*: Any set composed of $\overline{T}$ teams, should contain totally at most $m - \overline{M} - 1$ robots messengers, at any instant. This condition can be formalized as follows: $\sum_{k=1}^{\overline{T}} |T_k| \leq m - \overline{M} - 1$

The intuition underlying this condition (property C5) is the following: it guarantees that there still exists at least one correct robot messenger in the system if $\overline{T}$ simultaneous crashes occurred.

[3]this property is justified by the crash of the energy source of the pool.

4 Group Membership and View Synchrony algorithms

In this section, we study the problem of group membership and view synchrony in our system model, considering the three precedent failure models. The algorithms for models (A, B, C) are presented in the appendix.

For each failure model, we give a brief explanation and illustrate our algorithm by an example, then we give arguments showing its correctness, and finally we evaluate the energy and time required to run the algorithm.

We represent the system as a logical ring of nodes sorted by increasing order of teams identifiers, each node in the list represents a team of robots, the initial view contains all the teams in the system.

4.1 Group membership & failure-free (Model A)

We study the group membership and view synchrony in our system model, free of failures.

4.1.1 Description of the algorithm (Model A)

- *Condition*: The team initiator has at least one messenger in its pool.

We illustrate the group membership algorithm, in the case of failure-free by the following simple example:

Consider a system composed of three teams, we construct a logical ring of nodes $\{T_1, T_2, T_3\}$. The initial view is: $\{T_1, T_2, T_3\}$. We suppose that the team (T_2) starts to propose a new view (team initiator), and it invokes *join(T_p)* operation, the team T_3 invokes a *leave(T_3)* operation, and T_1 does not execute any operation.

The team T_2 starts the propose round by sending a messenger to T_3, the next team in the logical ring. The messenger transports a message which proposes the view: $v^i_{T_2} = \{T_1, T_2, T_3, T_p\}$.

When the team T_3 receives this message, it behaves as follows:

1. generates its own message: $T_3.leave(T_3)$

2. merges msg_{T_2} and msg_{T_3} then proposes the view $v^i_{T_3} = \{T_1, T_2, T_p\}$.

3. sends a messenger to T_1, with the new view $v^i_{T_3}$.

When the team T_1 receives the messenger, it behaves in the same previous manner with the difference that T_1 does not change any thing, it acknowledges the current proposed view, and sends a messenger to T_2, which terminates the propose round and starts the commit round.

T_2 starts the commit round by sending the message *commit*$\{T_1, T_2, T_p\}$ to T_3 which acknowledges the current view v^i and sends a messenger to the next team.

The algorithm terminates when T_2 receives back the commit message that it has sent, and the group membership algorithm is successfully terminated, such that the team T_p has joined the group and T_3 has left it, and the new view is $v^i = \{T_1, T_2, T_p\}$.

4.1.2 Correctness arguments (Model A)

In this model, both messengers and teams are correct, which guarantees the correctness of the communications between teams, so all messages sent by a team are correctly received by the destination team, and also proves the correct termination of this algorithm, since it is guaranteed that a messenger returns back to the initiator after both the propose and commit rounds.

We show that the three properties of the Group Membership protocol, are satisfied by our algorithm.

1. Validity: According to the algorithm A, the removal of a team from a view is impossible unless some team executes the operation leave(). Also, the existence of a new team in a view, is possible only by executing the operation join(new-team).

2. Agreement: The commit round of the algorithm, allows to circulate the same view v_{fin} to all the teams, so the new view is v_{fin}, and after the termination of the algorithm, any two teams install the same view.

3. Termination: When a team T_p executes join(T_q), this request is broadcasted to the teams that follow T_p in the logical ring during the propose round, and then all the teams in the system agree on join(T_q) via the commit round. So, eventually a view v' is installed such that $T_q \in v'$.

4.1.3 Behavior Evaluation (Model A)

The algorithm executes two rounds, propose and commit. Since there are no failures in this model, the messenger performs $2n$ hops between the teams, in order to define a new view. In the Model A, the energy and time required are equivalent to $O(2n)$.

4.2 Group membership & messengers failure (Model B)

We study the group membership and view synchrony in our system model, in presence of messengers failure.

4.2.1 Description of the algorithm (Model B)

- *Condition*: The team initiator has initially at least $(\overline{M} + 1)$ messengers in its pool.

In Model B, the team initiator executes the propose and commit rounds by sending a set of messengers which has at least one correct. We illustrate the algorithm by the same example of Model A:

The team T_2 starts the propose round by sending a set of $(\overline{M} + 1)$ messengers to T_3, such that each messenger carries the same message which proposes the view: $v^i_{T_2} = \{T_1, T_2, T_3, T_p\}$.

When the team T_3 receives this set of messengers (or at least one), it behaves as follows:

1. unifies all the identical messages received from T_2.

2. generates its own message: *T_3.leave(T_3)*.

3. merges msg_{T_2} and msg_{T_3}, then proposes the view $v^i_{T_3} = \{T_1, T_2, T_p\}$.

4. sends the set of messengers that it has received to T_1, with the new view $v^i_{T_3}$.

When T_1 receives the set of messengers from T_3, it does not change the view, acknowledges the current proposed view, and sends the messengers to T_2, which terminates the propose round and starts the commit round when it receives at least one messenger from T_1. T_2 starts the commit round by sending the same set of messengers with the message $commit\{T_1, T_2, T_p\}$ to T_3 which acknowledges the current view v^i and sends the set to T_1.

The algorithm terminates when T_2 receives back at least one messenger of this set carrying the commit message that it has sent, and the group membership algorithm is successfully terminated, such that the team T_p has joined the group and T_3 has left it, and the new view is $v^i = \{T_1, T_2, T_p\}$.

4.2.2 Correctness arguments (Model B)

The condition $|initiator| \geq (\overline{M} + 1)$ guarantees that the team initiator has at least **one correct** messenger, we show that this condition ensures the correct termination of the algorithm.

In the Model B, we need to send $(\overline{M} + 1)$ messengers from the initiator to the next team, in order to guarantee the correct reception of messages by the next team, [4] supposing that all the teams are correct. (assumptions of this failure model)

The cardinality of this set remains ≥ 1 and $\leq (\overline{M} + 1)$ because some messengers may crash before reaching their destinations, and this set of messengers is responsible of all the communications between the teams until return back to the initiator. (propose and commit rounds).

The algorithm guarantees that the same new view is acknowledged by all the teams in the system, since there is no team crash in this model.

The properties: Validity, Agreement, and Termination are discussed exactly as in the previous failure-free model.

[4]The set of messengers moving between teams may become smaller after each step of the algorithm, but this set is never empty.

4.2.3 Behavior Evaluation (Model B)

The initiator sends a set of $(\overline{M} + 1)$ messengers, and this same set performs $2n$ hops between the teams, so the energy consumed is of the order: $O(2(\overline{M} + 1)n)$. But the time required is the same as in the Model A, because the robots move simultaneously. So, the time required to run the algorithm is $O(2n)$.

4.3 Group membership & teams and messengers failure (Model C)

We study the algorithm of group membership in presence of both teams and messengers failure.

4.3.1 Description of the algorithm (Model C)

- *Condition 1*: The team initiator is a correct team.
- *Condition 2*: The team initiator has initially at least $(\overline{M} + 1)$ messengers in its pool.

We illustrate the group membership algorithm by the following simple example:

Consider a system composed of four teams, we construct a logical ring of nodes $\{T_1, T_2, T_3, T_4\}$. The initial view is: $\{T_1, T_2, T_3, T_4\}$. We suppose that the team T_2 starts to propose a new view (team initiator), and it invokes *join(T_p)* operation, for simplicity we suppose that other teams do not execute any operation, and the teams T_1 and T_3 are faulty.

propose round The team T_2 starts the propose round by sending a set composed of $(\overline{M} + 1)$ messengers to T_3, such that each messenger in the set carries the same message which proposes the view: $v^i_{T_2} = \{T_1, T_2, T_3, T_4, T_p\}$. When this set of messengers arrives to the site of T_3, it performs a crash detection protocol based on hand-shaking with all the workers of T_3. It confronts 2 cases:

- T_3 has crashed: the set of messengers returns back to T_2 indicating the crash of T_3, then the initiator changes the current view by removing T_3 from the group (forced leave) and sends this set of messengers to T_4 provided with the current proposed view $v^i_{T_2} = \{T_1, T_2, T_4, T_p\}$.

- T_3 is alive: it unifies the identical messages, and sends the set of messengers to T_4, as in Model B.

The propose round terminates when the team initiator receives back its set of messengers (or part of it).

commit round T_2 starts the commit round by sending the set of messengers with the message $commit\{T_1, T_2, T_4, T_p\}$ to T_4, which acknowledges the current view v^i and sends the set to T_1. When the messengers arrive to the site of T_1 they confront 2 cases:

- T_1 has crashed: the set of messengers returns back to T_2 indicating the crash of T_1, then the initiator changes the current view by removing T_1 from the group (forced leave) and restarts the commit round by sending this set of messengers to T_4 again, provided with the current commit view $v^i_{T_2} = \{T_2, T_4, T_p\}$. Then T_4 acknowledges the commit view and sends the messengers to T_2.

- T_1 is alive: it unifies the identical messages, and sends the set of messengers to T_2, as in Model B.

The algorithm terminates when T_2 receives back at least one messenger belongs to the set it has sent, provided with the commit message, and the group membership algorithm is successfully terminated, such that the team T_p has joined the group and (T_1, T_3) have left it because of their crashes, and the new view is $v^i = \{T_2, T_4, T_p\}$.

4.3.2 Correctness arguments (Model C)

In this model we have team failures in addition to messenger failures, so we need extra specifications concerning a team correctness.

We show that the two previous conditions guarantee that the algorithm terminates correctly. In our model a messenger can perform at most two hops, so when a messenger moves to a crashed team, the next hop should be to a correct one, else the messenger would be idle. The messenger returns to the team initiator after detecting a crashed team, and the initiator is a correct team according to (condition 1), while (condition 2) guarantees that the set of messengers sent by the initiator, has at least one correct messenger. This set performs all the hops between the teams as we discussed in the Model B.

In this model, we provide the system with a perfect failure detector, because the detection of a crashed team is carried out by a local hand-shaking mechanism, between at least one correct messenger and all the workers of the team. After detecting a crashed team by a messenger, this messenger moves to the team initiator (correct team), and proclaims the crashed team, consequently, the crash is detected correctly and deterministically.

The commit round, permits to provide each non-crashed-team with the most recent view, because the initiator restarts the commit round whenever it detects a crashed team, so the commit round terminates correctly by delivering the same view v_{fin} to all non-crashed teams.

The properties: Validity, Agreement, and Termination are discussed exactly as in the Model A.

4.3.3 Behavior Evaluation (Model C)

The set of $(\overline{M} + 1)$ messengers may perform additional hops because of teams failures, so this set needs to go backward to the initiator whenever it detects a crashed team. (additional $\overline{T}$ hops in the propose round), and $(n \cdot \overline{T}$ hops during the commit round). The energy consumed by messengers is calculated as follows:

Propose round(worst case):$(n + \overline{T})(\overline{M} + 1)$.

Commit round(worst case):$(n \cdot \overline{T})(\overline{M} + 1)$.

The total energy consumption is $(\overline{M} + 1)(\overline{T} + 1) \cdot n + \overline{T}(\overline{M} + 1)$.

The behavior evaluation in terms of energy can be written as : $O(\alpha \cdot n + \beta)$.

The behavior in terms of time is $(\overline{T} + 1) \cdot n + \overline{T}$, also it can be expressed as: $O(\lambda \cdot n + \mu)$.

Discussion The required energy and time to run the algorithm increase when fault tolerance requirements become harder. In the Model A the algorithm requires energy and time proportional to (2n), where n is the size of the system (group of teams). In Model B the energy becomes more significant than in Model A. It is $\overline{M}$ times greater, which is justified by the cost required to tolerate the messenger failures, but the execution time is equivalent to that in Model A. When the system is proned to team failures in addition to messenger failures in Model C, the required energy is $(\overline{M} \cdot \overline{T})$ times greater than that in the Model A, while the execution time is only $\overline{T}$ times greater.

5 Conclusion

We have introduced a distributed asynchronous system model composed of a group of teams of cooperative mobile robots. The teams in our model communicate by physical robot messengers. We have presented a group membership algorithm, discussed its correctness, and evaluated its behavior in terms of energy and time, in three possible failure models, the failure-free, the messenger failures, and both team & messenger failures models.

We have shown the conditions that should be satisfied to solve the problem of group membership in our system model in each different failure model. This technique of communications between teams of robots permits to implement a perfect failure detector since the detection of a crashed team takes place locally on its site. This property permits to solve many agreement problems in asynchronous distributed systems composed of a group of teams of robots.

Furthermore, in this model faulty robot messengers can be mapped to lossy channels in classical distributed systems. We guaranteed reliable communications in Model B by using one set of messengers that has at least a correct messenger, this set circulates the messages and supports all the communications between the teams of the system.

The model presented in our paper can be applied in situations where there are no established communications infrastructures for cooperative mobile robots.

In the future, we also intend to further investigate other distributed algorithms for cooperative autonomous mobile systems.

Acknowledgments

This research was conducted for the program "Fostering Talent in Emergent Research Fields" in Special Coordination Funds for Promoting Science and Technology by the Japan Ministry of Education, Culture, Sports, Science and Technology.

References

[1] T. Abdelzaher, A. Shaikh, F. Jahanian, and K. Shin. Rtcast: Lightweight multicast for real-time process groups. In *Proc. IEEE. Symp. on Real-Time Technology and Applications (RTAS'96)*, pages 250–259, Boston, MA, USA, 1996.

[2] B. Bellur, M. Lewis, and F. Templin. An ad-hoc network for teams of autonomous vehicles. In *Proc. Symp. on Autonomous Intelligent Networks and Systems (AINS-02)*, Los Angeles, USA, May 2002.

[3] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Comput. Surv*, 33(4):427–469, December 2001.

[4] P. Ezhilchelvan and R. de Lemos. A robust group membership algorithm for distributed real-time systems. In *Proc. 11th IEEE. Symp. on Real-Time Systems (RTSS '90)*, pages 173–181, Lake Buena Vista, Florida, USA, December 1990.

[5] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *ACM*, 32(2):374–382, April 1985.

[6] N. Hayashibara, X. Défago, R. Yared, and T. Katayama. The φ accrual failure detector. In *Proc. 23nd IEEE Intl. Symp. on Reliable Distributed Systems (SRDS'04)*, Florianópolis, Brazil, October 2004.

[7] H. Hu, I. Kelly, D. Keating, and D. Vinagre. Coordination of multiple mobile robots via communication. In *Proc. 13th IEEE Intl. Conf. on Mobile Robots (SPIE'98)*, pages 94–103, Boston, USA, November 1998.

[8] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev. Moshe: A group membership service for WANs. *ACM Trans. Comput. Syst.*, 20(3):191–238, August 2002.

[9] S. Schemmer and E. Nett. Managing dynamic groups of mobile systems. In *Proc. 6th IEEE Intl. Symp. on Autonomous Decentralized Systems (ISADS'03)*, pages 9–16, Pisa, Italy, April 2003.

[10] P. Urbán, I. Shnayderman, and A. Schiper. Comparison of failure detectors and group membership: Performance study of two atomic broadcast algorithms. In *Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN'03)*, pages 645–654, San Francisco, CA, USA, June 2003.

Algorithm of group membership & failure-free

In this section, we present our algorithm of group membership in the case of failure-free (Model A):

Algorithm(A): Group membership (failure-free)

```
Initial phase
1: $S \leftarrow \{T_1, T_2, \ldots, T_n\}$
2: $v_{ini} \leftarrow \{T_i, i \in [1..n]\}$
3: $v_{act} \leftarrow v_{ini}$
4: $old\_view \leftarrow v_{ini}$
5: $msg \leftarrow \emptyset$
6: operation = {join(new_team), leave(team)}
main algorithm ($T_i$)
7: if (operation = join (new_team)) then
8:     $T_i$.propose ($v_{act} \leftarrow v_{act} \bigcup \{new\_team\}$)
9: end if
10: if (operation = leave(team)) then
11:     $T_i$.propose ($v_{act} \leftarrow v_{act} \setminus \{team\}$)
12: end if
13: if ($T_i = initiator$) and ($|initiator| \geq 1$) then
14:     $msg.initiator \leftarrow initiator(ID)$
15:     $msg \leftarrow msg \bigcup \{v_{act}\}$
16:     send a messenger with (msg) to next($T_i$)
17:     wait until reception of the messenger sent
18:     when reception of the messenger sent
19:         begin-commit-round()
20:     end when
21: end if
22: if ($T_i \neq initiator$) then
23:     $msg \leftarrow msg \bigcup \{v_{act}\}$
24:     send the messenger received from previous($T_i$) with (msg) to next($T_i$)
25: end if
procedure begin-commit-round
26: if ($T_i = initiator$) then
27:     $v_{fin} \leftarrow v_{act}$
28:     $msg \leftarrow commit(v_{fin})$
29:     send the messenger received from previous($T_i$) with (msg) to next($T_i$)
30:     wait until reception of the messenger sent
31:     when reception of the messenger sent
32:         terminate-commit-round()
33:     end when
34: else
35:     send the messenger received from previous($T_i$) with (msg) to next($T_i$)
36: end if
end procedure
37: $new\_view \leftarrow v_{fin}$
```

Algorithm of group membership & messengers failure

We present in this section, our algorithm of group membership in presence of messenger failures (Model B):

Algorithm(B): Group membership (messengers failure)

```
Initial phase
1: $S \leftarrow \{T_1, T_2, \ldots, T_n\}$
2: $v_{ini} \leftarrow \{T_i, i \in [1..n]\}$
3: $v_{act} \leftarrow v_{ini}$
4: $old\_view \leftarrow v_{ini}$
5: $msg \leftarrow \emptyset$
6: operation = {join(new_team), leave(team)}
main algorithm ($T_i$)
7: if (operation = join (new_team)) then
8:     $T_i$.propose ($v_{act} \leftarrow v_{act} \bigcup \{new\_team\}$)
9: end if
10: if (operation = leave(team)) then
11:     $T_i$.propose ($v_{act} \leftarrow v_{act} \setminus \{team\}$)
12: end if
13: if ($T_i = initiator$) and ($|initiator| \geq (\overline{M} + 1)$) then
14:     $msg.initiator \leftarrow initiator(ID)$
15:     $msg \leftarrow msg \bigcup \{v_{act}\}$
16:     send set of ($\overline{M} + 1$) messengers provided with (msg) to next($T_i$)
17:     wait until reception of the messengers sent
18:     when reception of messengers sent
19:         begin-commit-round()
20:     end when
21: end if
22: if ($T_i \neq initiator$) then
23:     unify all the identical propose messages received from previous($T_i$) into one message (msg)
24:     $msg \leftarrow msg \bigcup \{v_{act}\}$
25:     send the set of messengers received from previous($T_i$) provided with (msg) to next($T_i$)
26: end if
procedure begin-commit-round
27: if ($T_i = initiator$) then
28:     $v_{fin} \leftarrow v_{act}$
29:     $msg \leftarrow commit(v_{fin})$
30:     send the set of messengers received from previous($T_i$) provided with (msg) to next($T_i$)
31:     wait until reception of the messengers sent
32:     when reception of messengers sent
33:         terminate-commit-round()
34:     end when
35: else
36:     unify all the identical commit messages received from previous($T_i$) into one message (msg)
37:     send the set of messengers received from previous($T_i$) provided with (msg) to next($T_i$)
38: end if
end procedure
39: $new\_view \leftarrow v_{fin}$
```

Algorithm group membership (teams and messengers failure)

In this section, we present our algorithm of group membership in presence of both teams and messengers failure (Model C):

Algorithm(C): Group membership (teams and messengers failure)

```
  Initial phase
 1: $S \leftarrow \{T_1, T_2, \dots, T_n\}$
 2: $v_{ini} \leftarrow \{T_i, i \in [1..n]\}$
 3: $v_{act} \leftarrow v_{ini}$
 4: $old\_view \leftarrow v_{ini}$
 5: $msg \leftarrow \emptyset$
 6: $list \leftarrow v_{ini}$
 7: operation = {join(new_team), leave(team)}
  main algorithm ($T_i$)
 8: if (operation = join (new_team)) then
 9:   $T_i$.propose ($v_{act} \leftarrow v_{act} \bigcup \{new\_team\}$)
10: end if
11: if (operation = leave(team)) then
12:   $T_i$.propose ($v_{act} \leftarrow v_{act} \setminus \{team\}$)
13: end if
  messenger:
  function detect_crash(messenger, team)
14: $detect\_crash \leftarrow TRUE$
15: for all robot-worker (w) ∈ team do
16:   if shake-hand(messenger, w) then
17:     $detect\_crash \leftarrow FALSE$        {the team is still alive}
18:     exit
19:   end if
20: end for
  end function
21: if detect_crash(messenger, $T_i$) then
22:   move to the team initiator.
23: end if
  Team:
24: if ($T_i = initiator$) and ($|initiator| \geq (\overline{M}+1)$) then
25:   $msg.initiator \leftarrow initiator(ID)$
26:   $msg \leftarrow msg \bigcup \{v_{act}\}$
27:   send set of ($\overline{M}+1$) messengers provided with (msg) to
      next($T_i$)
    Handling crashed teams(propose round)
28:   when reception of messengers carrying the failure de-
      tection message:$T_k$ has crashed and the current message
      is (msg)
29:     $v_{act} \leftarrow v_{act} \setminus \{T_k\}$
30:     $msg \leftarrow msg \bigcup \{v_{act}\}$
31:     if ($next(T_k) \neq initiator$) then
32:       send the received set of messengers carrying
          (msg) to next($T_k$)
33:     else
34:       begin-commit-round()
35:     end if
36:   end when
    end Handling crashed teams (propose round)
37:   wait until reception of messengers sent.
38:   when reception of messengers carrying a "non failure-
      detection message"
39:     begin-commit-round()
40:   end when
41: end if
42: if ($T_i \neq initiator$) then
43:   unify all the received identical propose messages into one
      message (msg).
44:   $msg \leftarrow msg \bigcup \{v_{act}\}$
45:   send the received set of messengers carrying (msg)
      to next($T_i$)
46: end if
  procedure begin-commit-round
47: if ($T_i = initiator$) then
48:   $v_{fin} \leftarrow v_{act}$
49:   $msg \leftarrow commit(v_{fin})$
50:   $list \leftarrow list \setminus \{detected\_crashed\_teams\}$ {$v_{fin}$ is identical
      to the updated logical ring of teams identifiers, so when a
      team receives the commit message, it discovers the next
      team to send.}
51:   if (the number of detected crashed teams < n-1) then
52:     send the received set of messengers carrying
        (msg) to next(initiator) in the new logical ring.
53:   else
54:     terminate-commit-round()           {the number of
        detected crashed teams = n-1, i.e. the initiator is the
        only correct team in the system}
55:   end if
    Handling crashed teams(commit-round)
56:   when reception of messengers carrying the failure de-
      tection message:$T_k$ has crashed and the current message
      is (msg)
57:     $v_{act} \leftarrow v_{act} \setminus \{T_k\}$
58:     $msg \leftarrow msg \bigcup \{v_{act}\}$
59:     begin-commit-round       {restart from the beginning}
60:   end when
    end Handling crashed teams (commit-round)
61:   wait until reception of messengers sent
62:   when reception of messengers carrying a "non failure-
      detection message"
63:     terminate-commit-round()
64:   end when
65: else
66:   unify all the received identical commit messages into one
      message (msg)
67:   send the received set of messengers carrying (msg)
      to next($T_i$) in the new logical ring.
68: end if
  end procedure
69: $new\_view \leftarrow v_{fin}$
```