In the Proceedings of the Workshop on Software Engineering and Mobility (co-located with IEEE/ACM ICSE 2001). Toronto, Ontario, Canada. May 13-14, 2001.

Reliable Communication for Mobile Agents with Mobile Groups

Flávio M. Assis Silva
LaSiD/DCC
Federal University of Bahia, Brazil
fassis@ufba.br

Raimundo J. A. Macêdo
LaSiD/DCC
Federal University of Bahia, Brazil
macedo@ufba.br

Abstract

A fundamental issue in the development of mobile agent systems is how to support reliability of agent-based applications. Among the reliability requirements of these applications is the need for coordinating reliably the actions of members of a group of mobile agents. We propose mobile groups as an approach to fulfilling part of this requirement. Mobile groups are an extension of the concept of process groups in traditional group communication systems that supports mobility of group members. In this paper we present the mobile groups model, informally define and illustrate some of their properties, and briefly show how it can be used in an example scenario.

1. Introduction

The mobile agent concept is being proposed to support different types of applications, including electronic commerce, workflow management, network management, and active networks, to cite some [1]. Mobile agents have been considered a concept that can be explored to provide benefits to distributed applications such as: better use of communication resources; flexible support for disconnected operation; flexibility for the management of software deployment and maintenance; among others [1, 2].

A fundamental issue in the development of mobile agent systems is how to support reliability of agent-based applications. Among the reliability requirements of these applications is the need for coordinating reliably the actions of members of a group of mobile agents. Agents belonging to a group, for example, should be able to reliably send messages to all the group members and to react consistently upon failures of some of the members.

One of the approaches for supporting consistent execution of a set of processes in distributed systems are *process groups* [3]. In a process group, processes communicate with each other by exchanging messages which are multicasted to the whole group. In order to preserve consistency among the state of group members, the group communication protocols guarantee certain properties such as atomic delivery (either all processes deliver a message or no one delivers it), message ordering guarantees (e.g., causal and total), and a sort of virtual synchrony where modifications on the group membership (caused by events such as process crashes and joins, etc.) are consistently ordered with respect to message delivery. In such an environment, operational processes perceive the same sequence of events, though, in reality, they may happen in an arbitrary order.

In traditional group systems, however, processes are essentially *static*. In order to provide guarantees to applications based on groups of agents similar to the ones provided by traditional group-based applications, we introduced the *mobile group* concept [4, 14]. A mobile group is an extension of the traditional concept of a process group that can directly support migrating processes as members of the group. With mobile groups, a migrating process has the ability to change its location in the distributed environment *while belonging to a group*. Mobile groups also provide message delivery guarantees and a sort of virtual synchrony. However, mobile groups provide these guarantees *despite the mobility of their members*. Furthermore, they make process mobility not only visible for the group, but also consistently ordered with other group actions (such as process crashes, joins, and leaves). An implementation of mobile groups by using conventional group systems is not satisfactory because process mobility would be hidden from the group actions history.

Due to the guarantees that they provide, mobile groups represent a suitable abstraction for the development of reliable agent-based applications.

After discussing related work in section 2, this paper presents some properties of mobile groups in section 3 and presents an example scenario where mobile groups could be used in section 4. The paper is concluded in section 5.

2. Related Work

In traditional process groups systems, such as Horus [5] and Newtop [6], processes are static, i.e., they remain at the same place in the distributed environment during its whole life time. Movement of a process in these systems could be implemented by making the moving process leave the group before the movement and join it again after the movement. The group communication system, however, recognizes these operations (leaving and joining the group) as two distinct operations for two different processes, since, when a process joins a group, it will be considered a new process. In mobile groups, a process can migrate *while belonging to the group.* The mobile group system recognizes the movement action of the process and considers movement as a single group operation. This allows a more efficient implementation of the group service and makes possible the synchronization of messages with relation to movement. Using traditional group communication protocols on top of a layer providing process migration transparency (as, for example, provided by systems such as Voyager [7]) would not provide a satisfactory solution, since process mobility would be hidden from the group service, making it cumbersome to implement some functionality such as synchronization of messages with relation to movement.

Previous work incorporated movement in group communication services for environments with mobile devices, for example: protocols for total or causal ordering and a membership service (e.g. [8, 9]). In these environments processes start and terminate their executions on the same host. However, since the host may be mobile, a process may change its (physical) location in the environment while the host where it is moves. In mobile groups we are considering that hosts are not mobile, but processes can move from a host to another. These problems are similar in the fact that a process can change its location in the distributed environment, but they differ in other aspects, such as scalability and in the way message routing is performed (at the application level, in the case of mobile agents, and at the network level, in the case of mobile devices). The algorithms proposed for group membership in mobile environments, however, do not tolerate host failures [8, 9] or would restrict the possibility of exploring locality if adapted to be used with mobile agents [9] (the algorithm in [9] is based on a static server with which the group members must communicate no matter where they are).

In the context of mobile agent systems, different forms of interaction between agents were proposed. Existing mobile agent systems (e.g., Voyager [7]) provide currently many forms of communication between agents (Remote Method Invocation, events, unreliable multicasts, etc.). In [10] the authors extend Linda to integrate mobility. Communication through tuple spaces and with guarantees of group systems are two different approaches for supporting mobile agent coordination that fulfill different set of requirements. In [11] a concept for coordinating groups of agents is described, but which is not fault tolerant. Approaches for reliable message delivery to mobile agents are proposed in [12] and [13]. In [12] the approach supports uni- and multicast, but failures are not tolerated. In [13] failures are considered, but the approach supports only unicast (peer-to-peer communication).

To the best of our knowledge no previous work exists that describes a concept for supporting guarantees such as the ones provided by traditional group communication systems to mobile agent based applications.

3. Mobile Groups

3.1. System Model and Assumptions

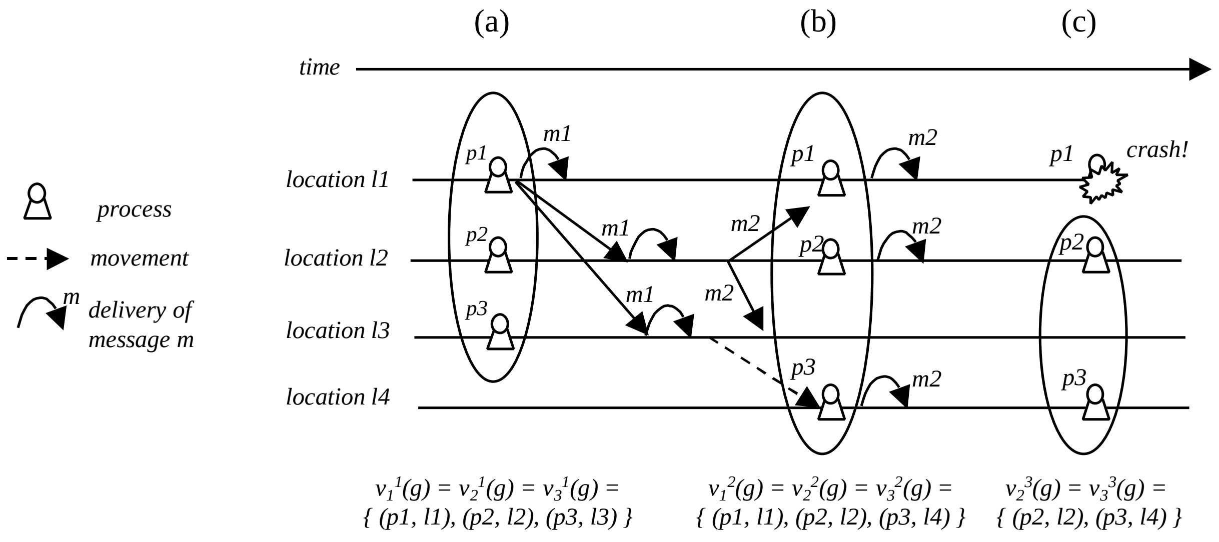
We assume a distributed system as a collection of mobile and static *processes, locations* and *communications channels*. A location represents a logical place in the distributed environment where processes execute. When a mobile process migrates, it moves from a location to another. Processes at different locations communicate by exchanging messages through the communications channels.

Let $L = \{ l_1, l_2, ..., l_n \}$ denote the set of all possible locations. Let P be the set of all possible processes. A mobile group is denoted by the set of processes $g = \{ p_1, p_2, ..., p_n \}$, g Í P. On a mobile group, four operations are defined:

- *join(g, p): issued* by process *p*, when it wants to join group *g* ;
- *leave(g, p)*: issued by process *p*, when it wants to leave group *g*;
- *move (g, p, l):* issued when a mobile process *p* wants to move from its current location to location *l*;
- *send(g, p, m)*: issued by process *p* when it wants to multicast a message to the members of group *g*.

We distinguish between a message being *received* at a location through the communication channel and a message being *delivered* to a process p. The delivery of a message to a process can be delayed to satisfy synchronization requirements enforced by the system.

A process *p* also *installs views*. In mobile groups a view $v(g) = \{ (p_1, l_1), (p_2, l_2), ..., (p_n, l_n)\}$, $p_i \in g$, $l_i \in L$, $\forall\ i$, is a mapping between processes of group g and locations. A view represents the set of group members which are mutually considered operational in a given instant of the group existence and indicates the locations where these members are (a pair *(p, l)* in a view indicates



(a)
(b)
(c)
time
process
movement
delivery of message m
m
location l1
location l2
location l3
location l4
p1
p2
p3
m1
m2
crash!
$v_1^1(g) = v_2^1(g) = v_3^1(g) =$ { (p1, l1), (p2, l2), (p3, l3) }
$v_1^2(g) = v_2^2(g) = v_3^2(g) =$ { (p1, l1), (p2, l2), (p3, l4) }
$v_2^3(g) = v_3^3(g) =$ { (p2, l2), (p3, l4) }

Figure 1: An example of mobile groups

that process *p* is currently at location *l*). This set can change dynamically on the occurrence of process crashes (suspicions) or when processes deliberately leaves, joins the group or move from a location to another. Every time a change occurs in the group view, a new view is installed at (operational) process members. Each view installed by a process is associated with a number which increases sequentially with group view installations. In a group $g=\{p_1, p_2, ..., p_n\}$, $v_i^j(g)$ denotes the view number *j* installed by process p_i.

Figure 1 illustrates a group initially with three process members, *p1, p2* and *p3.* These processes install, respectively, views $v_1^1(g)$, $v_2^1(g)$ and $v_3^1(g)$ (Fig. 1a). These views are equal and indicate that processes *p1, p2*, and *p3* consider one another operational and each process knows the current location of the others ($v_1^1(g)=v_2^1(g)=v_3^1(g)=\{(p1, l1), (p2, l2), (p3, l3)\}$. Later process *p2* moves to location *l4* (the movement is represented by the dashed line in Fig. 1b). A new view is installed by each process, reflecting that process *p3* is now at a new location, *l4* (views $v_1^2(g)=v_2^2(g)= v_3^2(g)=\{(p1,l1), (p2,l2), (p3,l4)\}$. Later, location *l1*, where *p1* is, crashes (Fig.1c). A new view will be installed by processes *p2* and *p3,* reflecting that, due to the failure, process *p1* was removed from the group (views $v_2^3(g)=v_3^3(g)=\{(p2, l2), (p3, l4)\}$).

We consider the communication channels to be reliable, i.e., message delivery in sequential order (FIFO order) is guaranteed. We assume that message transmission and processing times cannot be accurately estimated (i.e., an asynchronous system) and that processes fail only by crashing, i.e., by stopping functioning without producing any further action. We assume also that a majority of processes of a group view does not crash.

3.2. Mobile Group Properties

The consistency between views installed by processes in a group is guaranteed by a *membership protocol.* The main goal of this protocol is to ensure that each group member installs an identical sequence of views. Ensuring that each group member will install an identical sequence of views will provide the group members with a mutually consistent view of the group in relation to process crashes, joins, leaves *and movement operations.* Also, messages sent to the group will be synchronized in relation to all these events. Informally, operational processes in the group will see the same set of events of the group.

The complete set of properties of the mobile group membership and the membership protocol used to enforce them are formally presented in [4]. Below we describe informally some of these properties (illustrated in Fig.1):

View Safety Properties

- VSP1 (*Validity)* : only members of a group view install the corresponding view;
- VSP2 (*Unique Sequence of Views)*: there will be only one sequence of views for the group. If process p_i installs $v_i^k(g)$ and process p_j installs $v_j^k(g)$ then $v_i^k(g) = v_j^k(g)$, ∀ i, j, *k*. Observe in Figure 1 that views with the same number (upper index) are equal.

Message Delivery Properties

Safety Property

- MD1(Atomicity): any two member processes of *g* that install two consecutive views, deliver the same set of messages between them. Observe in Figure 1 that messages *m1* and *m2* where delivered by all processes at the same view (*m1* in view v_i^1 and *m2* in view v_i^2).

Message *m2*, for example, could not have been delivered by a process, say *p1*, at view v_1^1 and by another process, say, *p2*, at view v_2^2.

Liveness Property

- MD2(Liveness): if a process p_i sends a message *m* in view *r*, then provided it continues to function as a member of *g*, it will eventually deliver *m* in some view $v_i^{r'}$, $r' \geq r$. A message sent in a view might be delivered in a future view. This is illustrated in Figure 1, where message *m2* was sent in view v_2^1, but delivered in views v_i^2.

These properties enforce a virtual synchrony between processes, i.e., that processes in a group will install consistent views and will see the same set of messages between consecutive views.

4. An Example Scenario

In this section we describe a (typical) scenario where an application uses mobile agents to automate a process to reserve a flight seat on behalf of a user. Among other requirements (price range, seat class, etc.), the user imposes that, before a flight seat reservation is done, the prices of a minimum number of flight companies, say three, must be compared.

In general terms, this scenario could be implemented as follows. First three agents are created. Each will be in charge of determining the price of a flight ticket at a company that fulfills the user requirements. Having a priori a list of flight companies to visit, each agent chooses one of them and moves to its site to check that company's ticket price. If it does not succeed (for example, there is no more seats available in flights of the company), the agent moves to another company on the list and repeats the process until it succeeds.

The actions of the agents must be coordinated not only to compare the price offers found, but also to make the group react consistently on failures and exceptions. For example, if one of the agents fails, another one must be created to complete the set of three agents looking for price offers. Additionally, when an agent wants to visit a new company, it should choose a company that has not been previously visited by other agents in the group.

Mobile groups are a suitable abstraction to support the coordination of agents in this scenario and could be used as follows. The set of agents used in the scenario forms a mobile group. Each agent represents a process of the group. With the support provided by mobile groups each agent will have a consistent view of the configuration of the group. If an agent fails, the other agents will eventually know about this and will be able to act upon it. For each view a *coordinator* for the group can be defined, by applying a certain rule over the set of processes in the view (for example, the coordinator is the process with the greatest process id in the group, compared lexicographically). The coordinator will be the agent that will, for example, create new agents to do the job of failed ones. Also, since mobile groups provide a consistent view of process locations, when an agent decides to visit a new company, it will not move to locations it knows were already visited by another agent of the group.

More specifically, the execution of an agent could be (in general terms) described as follows. Views reveal agent movements, suspicions of agent crashes, and joins and leaves of agents to/from the group. Whenever a new view is installed by an agent, it can, for example, react as follows: if it verifies that a new agent joined the group, it multicasts its state (the set of previously visited companies and ticket prices it knows about) to inform new group members of previous group actions; if the view indicates that some group member has moved (an agent already in the group has changed its location information in the view), it adds the new location (extracted from the view) to its set of visited locations; if the view reveals that the agent itself has a new location attribute in the view, then it has moved and thus it interacts with a local server to try to obtain a ticket price. Additionally the agent will apply a previously combined function over the ids of the agents in the view to determine if it is the coordinator for that view or not.

The group system also signals a process if it is no longer a member of the group. In this case the process/agent exits.

When an agent succeeds in finding a price offer, it informs the user application. When the application acknowledges the receipt of its price information, the agent informs the group of this fact, by multicasting a message to the group. When a member of the group receives at least three such messages from members of the group, then the application has already at least three ticket prices. The agent then exits.

Periodically the coordinator of the view checks if there is less than three members in the group, by checking the number of members in the current view. If yes, it creates new agents to complement the group.

5. Conclusion

In this paper we have presented the mobile groups general model and informally some of the properties of a mobile group membership protocol for such groups. The concept supports a form of virtual synchrony for a set of mobile agents that provides a consistent view of events in the group, allowing the agents to act upon them in a coordinated form. Although mobile groups relate to a set of previously published works, we are not aware of some other concept that provides a similar functionality.

Mobile groups complement other efforts in providing reliability of mobile process based applications, such as concepts for mobile agent fault tolerance and transactional support [2].

Mobile groups are, however, a first step towards providing an effective support for coordinating groups of agents. Much effort still must be expended in further developing this concept, for example, for providing stronger message delivery guarantees and better support for partitioning.

6. References

[1] A.Fuggetta, G.P.Picco, G.Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering.* Vol.24, No.5. May, 1998

[2] F.M.Assis Silva. *A Transaction Model based on Mobile Agents.* PhD Thesis. Technical University Berlin. 1999

[3] K.Birman. The Process Group Approach to Reliable Distributed Computing. *Communications of the ACM*, Vol. 9, No. 12. pp. 36-53, December 1993.

[4] R.J.A.Macêdo, F.M.Assis Silva. Mobile Groups. *Technical Report RI001/01.* LaSiD/UFBA (Distributed Systems Laboratory / Federal University of Bahia). February, 2001. Submitted for publication.

[5] R.Renesse, K.Birman, R.Cooper, B.Glade, P.Stephenson. The Horus System. In K. Birman e R. Renesse, editors, *Reliable Distributed Computing with the Isis Toolkit*, pp. 133-147. IEEE Computer Society Press, Los Alamitos, CA, 1993.

[6] P.Ezhilchelvan, R.Macêdo, S.Shrivastava. Newtop: A Fault-Tolerant Group Communication Protocol. In *Proc. of the IEEE 15th Int. Conf. on Dist. Comp. Syst.* Vancouver, pp. 296-306, 1995.

[7] ObjectSpace. *Voyager – ORB 3.1 Developer Guide*. Object Space, Inc. 1999

[8] R.Prakash, R.Baldoni. Architecture for Group Communication in Mobile Systems. Proceedings of the 17th In *Procs. of the IEEE Symposium on Reliable Distributed Systems.* Indiana, USA. October, 1998

[9] A.Bartoli. Group-based Multicast and Dynamic Membership in Wireless Networks with Incomplete Spatial Coverage. *Mobile Networks and Applications*. Vol.3. Baltzer Science Publishers. 1998

[10] G.P.Picco, A.L.Murphy, G.-C.Roman. Linda Meets Mobility. In *Proceedings of the 21st International Conference on Software Engineering (ICSE'99),* Los Angeles (USA), D. Garlan and J. Kramer (eds.). ACM Press. May, 1999

[11] J.Baumann, N.Radouniklis. Agent Groups in Mobile Agent Systems. In *Procs of the Distributed Applications and Interoperable Systems (DAIS'97).* H.König, K.Geihs, T.Preuá (eds.). Chapman & Hall. 1997

[12] A.Murphy, G.P.Picco. Reliable Communication for Highly Mobile Agents. In *Proceedings of the Joint Symposium ASA/MA'99.* October 1999

[13] M.Ranganathan, M.Bednarek, D.Montgomery. A Reliable Message Delivery Protocol for Mobile Agents. In *Proceedings of the Joint Symposium ASA/MA2000*. September 2000

[14] F.M.Assis Silva, R.J.A.Macêdo. Reliability Requirements in Mobile Agent Systems. In *Proceedings of the II Worshop on Tests and Fault Tolerance.* Curitiba, Brazil. July, 2000.