Proceedings of the Fourth European Research Seminar on Advances in Distributed Systems (ERSADS'01). Bertinoro, Italy. May 2001. pp.209-214

Integrating Mobility into Groups

Raimundo J. A. Macêdo, Flávio M. Assis Silva

Distributed Systems Laboratory – LaSiD
Computing Science Department, Federal University of Bahia
Campus de Ondina, CEP: 40170-110, Salvador-BA, Brazil
{macedo, fassis}@ufba.br

Abstract

Some form of *migrating process*, i.e., a process that can change the node during its execution, is being frequently proposed as a basic component for designing distributed applications. Similarly to distributed applications based on static processes, applications based on processes that can migrate also need forms of reliable cooperation between processes. In order to fulfil part of this requirement, we present the concept of *mobile groups*. Analogously to traditional group systems [5,6,14,16,18], mobile groups also provide message delivery guarantees and a sort of virtual synchrony. However, mobile groups provide these guarantees *despite the mobility of its members*. Furthermore, they make process mobility not only visible for the group, but also consistently ordered with other group actions (such as crashes, joins, leaves, and other migrations). In this paper, we discuss the motivations for the mobile groups, define their properties, and outline a membership protocol for such groups.

1. Introduction

Process Groups have been widely used as a mechanism for supporting consistent execution of sets of co-operating processes in distributed systems [3,23]. In a process group, processes communicate with each other by exchanging messages which are multicast to the whole group. In order to preserve consistency, 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 a mutually consistent ordered sequence of events, though, in reality, they may happen in an arbitrary order.

In traditional group communication systems [5,6,14,16,18], after a member joins a group, it generally remains at the same location in the distributed environment (until it crashes or leaves the group). However, such stationary processes are no longer the unique way of structuring distributed applications. Some form of *migrating process*, i.e., a process that can change the node in which it is running, during its execution, is being frequently proposed as a basic component for designing distributed applications. An example of such migrating processes that have attracted the attention of researchers in the last years are the *mobile agents*. The mobile agent concept is being proposed to support different types of applications, including electronic commerce, workflow management, network management, and distributed information retrieval [1,2].

Similarly to distributed applications based on static processes, applications based on processes that can migrate also need forms of reliable co-operation between processes. In order to fulfil part of this requirement, we present the concept of *mobile groups*. Mobile groups are an extension of the traditional concept of groups that supports *moving processes* as members of a group. A moving process has the ability to change its location in the distributed environment while belonging to a group. Analogously to traditional group communication systems, mobile groups also provide message delivery guarantees and a sort of virtual synchrony. However, mobile groups provide these guarantees *despite the mobility of its members*. Furthermore, they make process mobility not only visible for the group, but also consistently ordered with other group actions (such as crashes, joins, leaves, and migrations). 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 mobile process or agent-based applications.

After discussing related work and motivation in section 2, the Mobile Groups and its main properties are presented in section 3. Section 4 discusses a typical scenario for the use of Mobile Groups and section 5

outlines a membership protocol which meets their properties. Finally, in section 6 some conclusions are drawn.

2. Related Work and Motivation

The movement or migration of a process in traditional group systems [5,6,14,16.18] could be mimicked 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 (usually, with a new identifier). 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 the movement as a single group operation. This allows a more efficient implementation of the group system and makes possible the synchronization of messages with relation to movement. Using traditional group communication protocols on top of a layer providing process migration (e.g., the Voyager system [7]) would not yield 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 location in the physical environment when the related host 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 (processes are expected to migrate more often) and in the way messages are routed (done at the application level, in the case of mobile agents, and at the network level, in the case of mobile devices). Thus, the directed adaptation of such algorithms to groups of mobile processes or agents is restricted (the algorithm in [9], for instance, 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 as those provided by traditional group communication systems in the context of mobile agents 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 mobile processes execute. When a mobile process migrates, it moves from a location to another. Processes at different locations communicate by exchanging messages through reliable communications channels. That is, transmitted messages are received uncorrupted and in the sequential (FIFO) sent order. No bounds on message transmission times or relative speeds of processes are assumed (i.e., asynchronous environment), and processes are assumed to fail only by crashing (without producing any further action).

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 $g = \{ p_1, p_2 , ..., p_n \}$, $g \acute{I} P$. 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*.

A process *p* also *installs views*. In mobile groups a view $v(g) = \{ (p, l) \mid p \hat{I} g \text{ and } l \hat{I} L\}$ 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 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 by a group membership protocol. Each view installed by a process is associated with a number which increases monotonically 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.

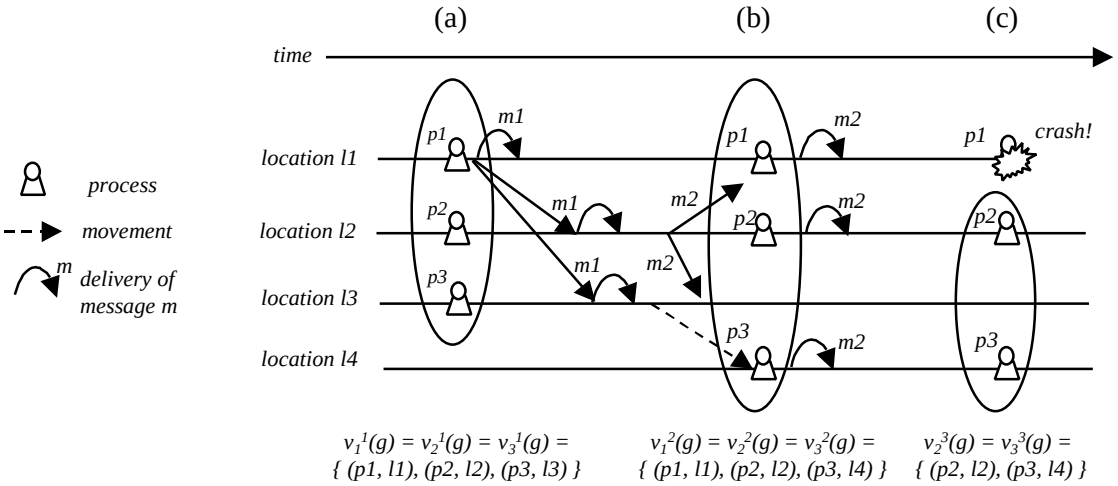

Figure 1: An example of mobile groups

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 identical 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 *p3* 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 $v_2^3(g)=v_3^3(g)=\{(p2, l2), (p3, l4)\}$).

3.2 Mobile Group Properties

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

View Safety Properties

- VSP1 (*Validity)* : only members of a group view install the corresponding view; that is, if a process $p_j \in g$ installs a view $v_j^i(g)$, then $p_j \in v_j^i(g)$.
- VSP2 (*Unique Sequence of Views)*: there will be only one sequence of views for the group. Formally, if a process $p_j \in v_j^i(g)$ installs $v_j^{i+1}(g)$ and a process $p_k \in v_j^i(g)$ also installs $v_k^{i+1}(g)$, then $v_j^{i+1}(g) = v_k^{i+1}(g)$.

Unique sequence of views is a necessary condition for the so-called primary component membership where only one component of the group is allowed to make progress. A common way of enforcing this behavior is to require that the majority of processes in a view v_j^i assent in the composition of view v_j^{i+1}. Observe in Figure 1 that views with the same number are identical.

Message Delivery Properties

If $g = \{p_1, p_2, ..., p_n\}$ is a mobile group, let *deliver(m, p_i, k)*, $p_i \in g$ denote the delivery of a message *m* to process p_i in view $v_i^k(g)$. We define the following safety and liveness 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* were delivered by all processes at the same view (*m1* in view v_i^1 and *m2* in view v_i^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_2^2.

MD1 and MD2 together enforce a virtual synchrony between processes. The actual semantics of the virtual synchrony enforced by MD1 and MD2 is similar to the one initially proposed by Birman [14] and formalized in [17]. Birman's virtual synchrony requires that $r = r'$ in MD2 and can be implemented by blocking sending messages during the view installation procedure. Our definition is more related to the ones of Transis [18] and Newtop [6] systems which differ from ours by the fact they were primarily intended to partitionable memberships

4. An Example Scenario

In this section we briefly describe a (typical) scenario where an application uses mobile agents to make flight seat reservations. Before a flight seat reservation is done by the user of the application, 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 ticket at a company that fulfills the user requirements. Having a priori a list of flight companies to visit, each agent chooses one of the companies on the list and moves to its site to check the ticket's price at that company. 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. For example, if one of the agents fails, another one must be created to complete the set of three agents. Additionally, the agents should choose companies to visit that have 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 identifier 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.

5. A Membership Protocol for the Mobile Groups

When a mobile group *g* is created, every group member p_k installs an initial view $v_k^1 = \{ (p_1, l_1), (p_2, l_2) , ..., (p_n, l_n)\}$. After the initial view is installed, any modification on the structure of the mobile group (migrations, addition or deletion of members) will result in new views being installed, forming the sequence $v_k^1, v_k^2, ..., v_k^m$ where *m* represents a given moment on the view evolvement history. These view changes are carried out by a membership protocol such as the one we outline below (complete description of this protocol is found elsewhere[4]).

Our group membership protocol is based on a consensus module in order to reach agreement on new views to be installed by operational group members. The consensus module we use is based on the unreliable failure detector concept, proposed and investigated by Chandra, Hadzilacos and Toueg [19, 22]. Thus, we assume in our system model the existence of a distributed ◊S failure detector. Furthermore, we also assume that a majority of processes of a group view does not crash as required by the ◊S Consensus protocol.

A process sends a message *m* to a group *g* by using a best-effort multicast primitive, denoted mcast(m,g), which causes the delivery of m to the current membership of g (denoted *m.g*) as long as the *m* sender process does crash.

Handling Process Crashes

When *m* is received by a destination process *p*, *p* $\hat{I}$ *m.g*, *m* is immediately delivered to *p* and stored in a local buffer until *m* is known to be stable (i.e., received by all processes in m.g). If a message remains unstable for too long, the membership service will start a new view installation procedure for removing possibly crashed processes from the current membership view and delivering the unstable messages not yet delivered. This procedure, which is carried out through a ◊S based consensus protocol, guarantees that all operational members deliver the same set of messages and install the same sequence of views.

During view installation, all the processes are required to send their sets of unstable messages and the new view will be formed removing those processes which were suspected by the local failure detector, provided that a majority of the processes (whether suspected or not) have sent the unstable sets. Thus, the new view installation will only progress if any of the operational group members fail in sending the corresponding unstable set (notice that a real crashed process that failed in sending the acknowledgement to a given message that is unstable, will also fail in sending the unstable set). Such processes –those which did not send the unstable set- are considered as crashed and a new view will be installed which does not include them. We should bear in mind, however, that those suspected processes may be just too slow (due to a overloaded site or connection link, for instance). Although a false suspicion[1] may cause the removal of a operational process from a group, the membership service will do this removal in such a way that the membership information will always be kept mutually consistent among all not suspected processes (thanks to the consensus module).

Though we present here a novel protocol (since its primary intended to Mobiles Groups), a similar use of multiple consensus executions for solving agreement problems (such as membership and atomic broadcast) have been previously presented in other works [22,15,20].

Handling Moves

A process p_i of a group g willing to move to a new location l, must issue the operation move(g,p_i,l). As a result, the mobile group service will first make sure that there are conditions to the required migration. If so, the location information of p_i is updated and a new consensus is started to try to install the new view with the new location for p_i. This consensus procedure is requited since all the events (crash, joins, leaves, migrations, and message delivery) should be ordered with respect to the new process location installation.

6. Conclusions

In this paper we presented the properties of the mobile groups and outlined a membership protocol that satisfies the defined properties. Mobile groups support a form of virtual synchrony in which messages are synchronized with not only crashes (suspicious), leaves and join operations, but also with movement operations. Therefore, mobile groups fulfill part of the requirements that arise in current distributed applications such as mobile agent based applications, such as the one outlined in section 4.

To the best of our knowledge this is the first work that provides and formally defines a concept of group which supports moving processes by integrating the movement events inside the group communication protocols. By doing that, we were able to enforce semantics for synchronizing the delivery of messages with relation to movements, and, as a consequence, we can extend the functionality of the system. For instance, by defining alternative semantics for message delivery that recognizes a moving process (for example, total

[1] Timeout value should be carefully chosen in order make false suspicions rare.

order delivery with respect to moving actions). That would not be achieved satisfactorily by using traditional group communication systems.

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 support for partitioning.

7. 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. Tech. Report RI001/01. LaSiD/UFBA (Distributed Systems Laboratory / Federal University of Bahia). February, 2001.

[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. Proc. of the 17th IEEE Symp. on Reliable Dist. Systems. Indiana, USA. Oct, 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. Proc. of the 21st Int. Conf. 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. 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. 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. Proc. of the Joint Symposium ASA/MA2000. September 2000

[14] K.Birman, A.Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. ACM Transactions on Computer Systems, 9(3):272-314, August 1991.

[15] R.Guerraoui and A.Schiper. Consensus service: A modular approach for building fault-tolerant agreement protocols in distributed systems. Proc. of the 26th Int. Symp. on Fault-Tolerant Computing (FTCS-26), pages 168-177, Japan, June, 1996.

[16] O.Babaoglu, M.Baker, R.Davali, and L.Gianchini, "Relacs: A communication Infrastructure for Constructing Reliable Applications in Large-Scale Distributed Systems". BROADCAST Project deliverable report, October 1994.

[17] E.Anceaume, B.Charron-Bost, P.Minet, and S.Toueg, On the formal specification of group membership services. Tech. Report 95-1534, Cornell Univ., Ithaca, USA, 1995.

[18] Y.Amir, D.Dolev, S.Kramer, D.Malki, Transis: A Communication Subsystem for High Availability. In Proc. of the 22nd Int. Symp. on Fault-Tolerant Comp. pp. 76-84, Boston, July, 1992.

[19] T.Chandra, V.Hadzilacos and S.Toueg, The Weakest Failure Detector for Solving Consensus. Journal of the ACM, 43(4):685--722, July 1996.

[20] M.Hurfin, R.Macêdo, M.Raynal,.F.Tronel. A General Framework to Solve Agreement Problems. Proc. of the IEEE Int. Symp. on Reliable Distributed Systems, SRDS'99, Lausanne. 1999.

[22] T.Chandra and S.Toueg, Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM, 43(2):225-267, March 1996.

[23] P.Veríssimo and L.Rodrigues. Group Orientation: a Paradigm for Modern Distributed Systems. In Proc. of the 5th ACM SIGOPS European Workshop, Mont Saint-Michel, France, September 1992.