Non-Blocking Transactional Mobile Agent Execution

Stefan Pleisch
IBM Research
Zurich Research Laboratory
CH-8803 Rüschlikon
spl@zurich.ibm.com

André Schiper
Distributed Systems Laboratory
Swiss Federal Inst. of Technology (EPFL)
CH-1015 Lausanne
Andre.Schiper@epfl.ch

1 Introduction

Mobile agents are computer programs that act autonomously on behalf of a user and travel through a network of heterogeneous machines. On each machine i, a place p_i $(0 \leq i \leq n)$ provides the logical execution environment for the agent at stage S_i of the agent execution. Agent a_i at the corresponding stage S_i represents the agent a that has executed the stage actions sa_j on places p_j $(j < i)$ and is about to execute on place p_i (Fig. 1).

To enable mobile agent technology for e-business, transaction support needs to be provided, in particular execution atomicity. Execution atomicity ensures that either all operations of the agent succeed, or none at all, and needs to be ensured also in the face of *infrastructure failures* (i.e., crash failures of agents, places, and machines). Note that a crashing machine leads to the crash of all places and all agents running on it, and a failure of a place also crashes the agents running on this place [3]. Indeed, any component in a system is subject to failures. In this context, we distinguish between *blocking* and *non-blocking* solutions for transactional mobile agents, i.e., mobile agents, that execute as a transaction. Blocking occurs, if the failure of a single component prevents the agent from continuing its execution. In contrast, the non-blocking property ensures that the mobile agent execution can make progress any time, despite of failures. While other approaches [5] block if the place running the mobile agent fails, the approach presented in this paper is non-blocking. A non-blocking transactional mobile agent execution has the important advantage, that it can make progress despite failures. In a blocking agent execution, progress is only possible when the failed component has recovered. Until then, the acquired locks cannot be freed (unless compensating transactions are assumed as in [1]). As no other transactional mobile agents can acquire the lock, overall system throughput is dramatically reduced.

2 The Problem of Execution Atomicity

An *atomic* mobile agent execution ensures that either all stage operations succeed or none at all. Assume, for instance, a mobile agent that books a flight to New York, books a hotel room there, and rents a car. Clearly, the use of the hotel room and the car in New York is limited if no flight to New York is available any more. In this case, we speak of a *semantic failure*. More generally, a semantic failure occurs if a requested service is not delivered because of the application logic or because the process providing this requested service has failed. On the other hand, the flight is not of great use if neither a hotel room nor a rental car is available. This example illustrates that either all three operations (i.e., flight ticket purchase, hotel room booking, and car rental) need to succeed or none at all, i.e., the operations have to be executed *atomically*.

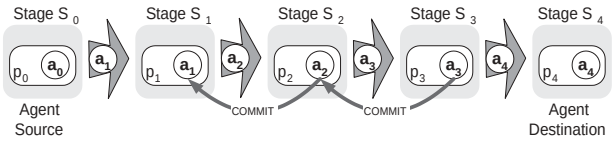

Figure 1. Successful transactional mobile agent execution T_a.

To achieve execution atomicity in a transactional mobile agent execution T_a, place p_i only forwards agent a_i to the place p_{i+1} at stage S_{i+1} if the execution of a_i has been successful (Fig. 1, with $n = 4$). At stage S_{n-1}, the successful execution of a_{n-1} triggers a commit to all places p_j $(0 < j \leq n-1)$. Consequently, only place p_{n-1} can decide to commit T_a. However, every place p_i can decide to unilaterally abort T_a, when it is executing a_i. Terminating T_a already at stage S_{n-1} instead of S_n prevents blocking due to disconnections of mobile devices (i.e., place p_n). The agent a_n is then kept at place p_{n-1} until p_n reconnects and is able to collect the result.

Infrastructure failures may lead to blocking or to a violation of the atomic execution of the transactional mobile agent. Assume that place p_2 fails while executing a_2 (Fig. 1). In an asynchronous system, where no bounds on communication delays nor on relative processor speed exist, p_0 and p_1 are left with the uncertainty of whether p_2 has actually failed or is just slow. In addition, it is impossible

for p_0 and p_1 to detect the exact point where p_2 failed in its execution. More specifically, they cannot detect whether p_2 has succeeded in forwarding the agent to the next stage or not. If p_2 has failed before forwarding a_2, the agent execution is blocked. To prevent blocking, another place such as p_1 could monitor place p_2. If it detects the failure of p_2, it could then abort T_a. However, unreliable failure detection potentially leads to a violation of the atomicity property. Indeed, assume that p_1 detects the failure of a_2. Place p_1 thus assumes the responsibility for the decision and decides to abort transaction T_a. However, because of unreliable failure detection, p_1 may erroneously suspect p_2. Actually, even if p_2 has failed, it may have succeeded in forwarding the agent to p_3, resulting in potentially conflicting decisions on the outcome of the transaction. Indeed, while p_1 decides to abort the transaction, p_3 may decide to commit T_a. This conflicting outcome clearly violates the atomic execution property of the transaction's operations, as certain operations are aborted (i.e., on p_1), whereas others are committed (on p_3) or may be committed later (on p_2).

3 Non-Blocking Transactional Mobile Agents

The approach we advocate adapts earlier work on fault-tolerant mobile agent execution [3] to provide non-blocking to the transactional mobile agents. Instead of sending the agent from one place to the other, it is sent to the set $\mathcal{M}_i$ of places p_i^j at stage S_i. This redundancy enables the mobile agent execution to proceed despite infrastructure failures, i.e., prevents blocking. As we do not assume reliable failure detection, redundant agents potentially lead to multiple executions of the agent code. The solution presented in [3] consists, for all agent replicas at stage S_i, to agree on (1) the place p_i^{prim} that has executed the agent, (2) the resulting agent a_{i+1}, and (3) the set of places of the next stage $\mathcal{M}_{i+1}$. In the context of fault-tolerant mobile agent execution, (1), (2), and (3) are important to prevent multiple executions of the agent, i.e., ensure the exactly-once property. All the places that have potentially started executing a_i, except p_i^{prim}, abort. Only p_i^{prim} commits the modifications of a_i (Fig. 2). In fault-tolerant mobile agent execution, a_i decides unilaterally which subtransaction to commit. More specifically, the decision is taken independently of the parent transaction, as no such transaction exists.

To achieve non-blocking atomic commitment, the modifications of a_i on the primary $p_i^{prim} \neq p_{n-1}^{prim}$ (i.e., the subtransaction sa_i^{prim}) are not immediately committed after the stage execution. In other words, stage action $sa_i^{prim} \neq sa_{n-1}^{prim}$ cannot unilaterally decide commit. This is fundamentally different from the fault-tolerant mobile agent execution approach in [3]. The decision to commit rather depends on the outcome of the transaction T_a.

To terminate a pending transactional mobile agent execution such as T_a, each primary place runs a *stationary* (i.e., not mobile) *stage action termination (SAT)* agent. During its execution, agent a maintains a list of all the SAT agents that it needs to contact in order to commit or abort the transaction. At every primary place p_i^{prim} ready to commit, a new entry is appended to this list. As soon as the outcome of T_a is clear, all SAT agents in the list are notified of this outcome. It is important that this message eventually arrives at all destination (i.e., the SAT agents participating in T_a). Indeed, a destination SAT agent that does not receive the decision message does not learn the outcome of the transaction T_a and still retains all locks on the data items. Hence, the decision message is distributed using a reliable broadcast mechanism that ensures the eventual arrival of the message at all destinations.

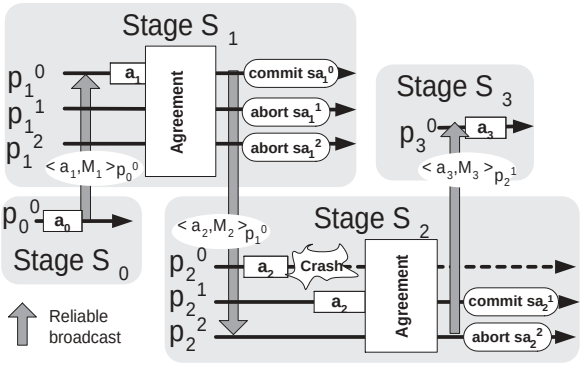

Figure 2. Non-blocking transactional mobile agent with crash of p_2^0.

The other ACID properties of a transactional mobile agent execution (i.e., consistency, isolation, and durability) are enforced using standard approaches [2], as well as for deadlock resolution, where we apply the timeout-based approach.

We have implemented a prototype system and evaluated its performance. The interested reader is referred to [4] for more details.

References

[1] F. Assis Silva and R. Popescu-Zeletin. Mobile agent-based transactions in open environments. *IEICE Trans. Commun.*, E83-B(5), May 2000.

[2] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, CA, USA, 1993.

[3] S. Pleisch and A. Schiper. FATOMAS: A fault-tolerant mobile agent system based on the agent-dependent approach. In *Proc. of Int. Conference on Dependable Systems and Networks (DSN'01)*, pages 215–224, Goteborg, Sweden, July 2001.

[4] S. Pleisch and A. Schiper. TranSuMA: Non-blocking transaction support for mobile agent execution. Technical Report rz 3386, IBM Research, 2001.

[5] R. Sher, Y. Aridor, and O. Etzion. Mobile transactional agents. In *Proc. of Int. Conference on Distributed Computing Systems (ICDCS'01)*, pages 73–80, Apr. 2001.