Ensuring Applicative Causal Ordering in Autonomous Mobile Computing

Samuel Quaireau[1] and Philippe Laumay[1,2]

[1] Laboratoire Sirac***
[2] Schlumberger-CP8
INRIA Rhône-Alpes ; 655, Avenue de l'Europe ; 38334 Saint-Ismier Cedex ; France
Tel : +33 4 76 61 53 89 - Fax : +33 4 76 61 52 52
Philippe.Laumay@inrialpes.fr

Abstract. There is a growing trend in developing applications for mobile computing systems in which mobile hosts can directly communicate disconnected from the wired network. In such heterogenous distributed applications, the Message Oriented Middleware (MOM) technology provides many solutions, such as causal message ordering to reduce the non-determinism. Usual implementations of causal ordering assume that it has to be ensured between all distributed processes, what goes with an unnecessary cost overrun. Moreover, the developer cannot adapt the causal ordering property to the semantic needs of a distributed application. We present a solution called *applicative causality* to ensure causal ordering in autonomous mobile environments, based on the knowledge of the application architecture and the communication patterns between processes, and allowing the developer to relax causality according its necessity in a semantic point of view.

1 Introduction

The will to include applications into mobile platforms leads the developers to design highly distributed architectures. Although mobile networks, with support stations and mobile nodes, represent a major portion of current mobile systems architectures, it is now clear that a new trend favoring direct interactions among mobile nodes is rapidly emerging with autonomous platforms like PDAs or Bluetooth equipments.

In this way, the middleware technology provides many solutions allowing to hide the handling of the distribution of services and computations to the developers. The MOMs (Message-Oriented Middleware [1]) use asynchronous message sending and are particulary adapted for mobile platforms thanks to features of message queuing (which ensures a certain reliability) or message ordering (which provides a way to reduce the non-determinism) for instance.

A common ordering mechanism uses logical time [2] to order events according to the causal order [3]. The causal precedence relation induces a partial order on the events of a distributed computation. It is a powerful concept, which helps to solve a variety of problems in distributed systems like algorithms design, concurrency measurement, tracking of dependent events and observation.

The causal order is often implemented with matrix clocks which pose scalability problem. Indeed, the linear increase of the number of processes causes a quadratic increase of the causal order handling cost ($O(n^2)$ for n processes). This issue becomes critical as the number of mobile equipments increase. Another problem especially true with autonomous mobile equipments is known as "false causality" [4]. This characteristic of causality induces unnecessary delayed delivery of messages if this causal precedence has no semantic meaning.

This paper proposes a solution based on *applicative causality*, which takes into account the semantic of the application to causally order the messages between processes. Our solution uses the AAA MOM[3] as a framework.

*** Sirac is a joint laboratory of *Institut National Polytechnique de Grenoble*, *INRIA* and *Université Joseph Fourier*. http://sirac.inrialpes.fr

[3] The AAA MOM was developed in the Sirac laboratory in collaboration with the *Bull-INRIA Dyade* consortium, and is freely available on the Web with an implementation of the JMS interface. http://www.objectweb.org/joram/

This paper is organized as follows. Related work on causality in mobile environments is surveyed in section 2. Section 3 shows our motivations guided by the AAA MOM (presented in section 3.1) which has led us to propose the *applicative causality* in section 4. We conclude in section 5.

2 Related work

Many solutions have been proposed to offer causal ordering on distributed application for mobile computing (a summary can be found in [5]). Although these solutions are well suited to mobile platforms like cellular phones (which must communicate through a *Mobile Support Station*) or smart cards, they are not appropriate for autonomous mobile platforms like PDAs which can communicate together in disconnected mode.

C.Baquero [6] presents a new way to causally order messages between a set of unlimited autonomous processes. But this solution requires a huge amount of information to allow a local characterization of the partial ordering of any two events, which limits the scalability.

In [7], the authors start with an experiment of 76 distributed processes exchanging messages. With a classical matrix clock the cost would have been $O(n^2) = k \times 76^2$; using "plausible clocks" to send information to only 5 or 6 well chosen processes, the causal order was respected in 9 cases out of 10. This paper shows the problem of data excess for a matrix clock which contains all potential information for the respect of causal order, even if some processes will never communicate together.

The authors of [4] raise the problem of "false causality": if a message arrives too early at a site, it will wait until all previous messages (in a causal meaning) are delivered, even though this causal precedence has no semantic meaning.

From this brief survey, we may conclude that the cost for causal order implementation could be reduced if we could know which processes communicate together instead of taking into account all the possible cases. The next section presents our motivations guided by the AAA MOM features.

3 Motivations

3.1 The AAA Environment

The AAA (Agents Anytime Anywhere) MOM [8] is a fault-tolerant platform that combines asynchronous message communication with a programming model using distributed, persistent software entities called agents[4]. Agents are autonomous reactive objects executing concurrently, and communicating through an "event → reaction" pattern [9]. Agents are persistent and their reaction is atomic, allowing recovery in case of node failure. The Message Bus guarantees the reliable, causal delivery of messages. The MOM is represented by a set of Agents Servers[5] (or servers for short) organized in a bus architecture (see figure 1).

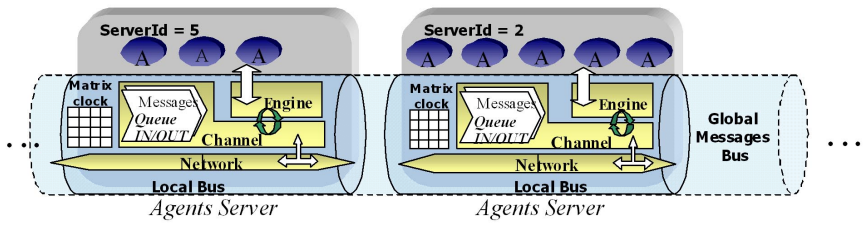


Fig. 1. Two interconnected Agent Servers details.

[4] Agents can be compared to the clients of the Message Bus (i.e. the MOM) and could be seen as "processes".

[5] Agents Servers could be considered as execution nodes

The combination of the agents and bus properties provides a solution to transient nodes or network failures, while causal ordering decreases the non-determinism of asynchronous exchanges. Each server is made up of three components, the *Engine*, the *Channel* and the *Network*, which are implemented in the Java language. The *Engine* guarantees the agents properties and the *Channel* ensures reliable messages delivery and causal order between servers. The causal order algorithm uses a matrix clock on each server, which has a size of n^2 for n servers.

This causal ordering is well suited to the bus architecture of AAA MOM, where all agents are "directly" connected and may communicate together. But in distributed applications, all processes rarely communicate with all others, especially in autonomous mobile environment. The causal ordering information added to messages should be restricted to the process (directly or indirectly) concerned by this messages.

This issue lead us to propose in section 4.1 a new approach of causal ordering based on the application topology knowledge (i.e. which agents communicate with others) called *applicative causality*.

3.2 Knowledge of application architecture

Past research around Architecture Definition Language (ADL [10]) has led to the definition of a toolkit for the design of AAA-based applications [11]. The global structure of an agent-based application is described with an ADL. The architecture description can be annotated with placement statements or properties for individual agents or groups of agents. For the description, the developer can use a *Graphic Construction Tool* to interconnect the agents and to set their properties. From this description, appropriate tools can generate a sequence of configuration actions to be performed by a *deployment service* in order to automate the application deployment.

Thanks to the tools, the developer adds the semantic to the application. The configuration obtained allows to know communication flows between the components and to adapt the MOM properties to the application topology. Thus, it is possible to use this knowledge for the causal ordering.

4 Proposal

4.1 Applicative causality

Once an application architecture is described, all software components involved in the application are known as well as all potential communication among them. Such a *configuration* of the application can be used to know before the deployment step which agents would have to send notifications, and to which ones. Moreover, it can be seen as a directed graph in which vertexes are mapping distributed processes (for example AAA agents), and directed edges are representing directional communications channel between them (cf. figure 2).

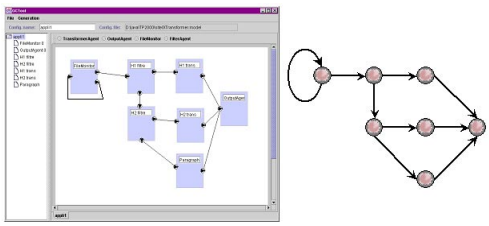
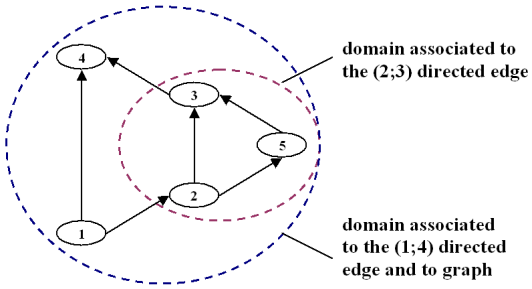


Fig. 2. An example of *configuration* and its associated directed graph

The operating principle of *applicative causality* is to group into domains some processes, for which shared messages must have to be causally ordered. These domains are built from the directed graph mapping the *configuration* of the application, according to the following definition:



Definition *The* domain associated to a (p, q) directed edge *is the* $D(p, q)$ *processes set containing* p, q *and all others processes* r *such that there exists an elementary (directed) path from* p *to* q *including* r. *The domain associated to the* (p, q) *directed edge is* associated to the graph *if there is no* (r, s) *directed edge such that* $D(r, s)$ *strictly contains* $D(p, q)$.

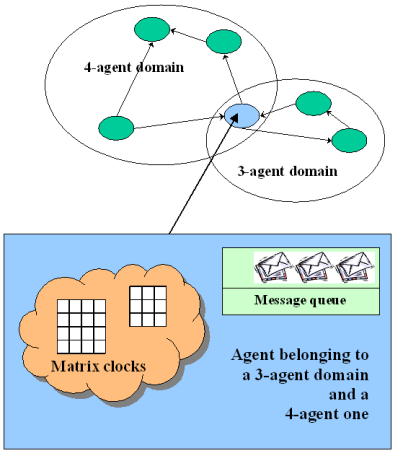
Theorem *Local causal ordering in each domain associated to the graph guarantees global causal ordering* (the proof can be found in [12]).

Within the AAA environment, these domains can be automatically computed from the application architecture. This is done in two steps: 1. construction of the directed graph from the knowledge of software components interconnections in the distributed application, then 2. computation of causality domains from the directed graph.

4.2 Runtime architecture

At the runtime level, the AAA MOM and agents have been defined as configurable elements. The result of the domain computation enables the configuration of causality in the agents just before the deployment step; that is to say each agent will be initialized by setting all matrix clocks according to the domains they belong to (see opposite figure).

Then the agents have to handle causal ordering in each domain. This is done in a transparent way for the developer by the middleware infrastructure, in a class defining the common behaviour for all agents and handling the causal local ordering. When a message is sent, it is stamped with matrix clock(s) according to the sender and receiver common domains. At the receipt of a message, this one will be either immediately delivered or stored in a message queue associated to the receiver agent, according to its stamp and receiver matrix clock(s).



4.3 Handling of false causality

A strong point of *applicative causality* is to allow a developer to relax causality from AAA development tools in order to reduce "false causality" [4]. This is particularly important to avoid that the delivery of some messages being unnecessary deferred to respect causal order, for example in direct communications between disconnected mobile platforms. Indeed, while connecting software components, the developer can specify that causal ordered communications between some processes has no sense from a semantic point of view, which means not considering corresponding directed edges when building the graph image of the application topology, and so relaxing causality where it is not needed.

5 Conclusion and future work

In this paper, we have presented *applicative causality*, a way to causally order messages in a distributed environment composed of autonomous platforms. Our proposal is based on the knowledge of the *architecture definition* of the application, allowing an handling of the causal ordering limited to the distributed processes led to communicate together. Consequently, this solution should

be scalable for many applications, and several research ideas are given in [12] in the other cases. Moreover, *applicative causality* allows the developer to reduce "false causality" while building the application, by relaxing the causal ordering property between processes for which it is not necessary. In one word, our solution allows to adapt during the construction of an application a MOM property (here the causal ordering) according to the developer needs, instead of using a costly global property imposed by the system.

Prospects for future work are first to use *applicative causality* in a mechanism of process migration (for load-balancing for example) in order to ensure the causal ordering between migrating processes. In the long run, use it to compute domains in a less static way, in order to allow mobile platforms to dynamically join or leave a distributed application.

Acknowledgements

The authors would like to thank Luc Bellissard for his precious comments on earlier versions of this paper, and Renaud Lachaize for his reviews.

References

1. G. Banavar, T. Chandra, R. Strom, and D. Sturman. A Case for Message Oriented Middleware. In *Lecture Notes in Computer Science*, volume 1693, pages 1–18. Distributed Computing 13th International Symposium, September 1999. ISBN 3-540-66531-5.
2. L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System, 1978.
3. K. Birman, A. Schiper, and P. Stephenson. Lightweight Causal and Atomic Group Multicast. In *ACM Transactions on Computer Systems*, volume 9, pages 272–314, August 1991.
4. P. Gambhire and A. D. Kshemkalyani. Reducing False Causality in Causal Message Ordering. *Lecture Notes in Computer Science - No. 1970*, pages 61–72, 2000.
5. C. Skawratananond, N. Mittal, and V. K. Garg. Lightweight Algorithm for Causal Message Ordering in Mobile Computing Systems. Technical Report TR-PDS-1998-011, The University of Texas at Austin, 1998. available via ftp or WWW at maple.ece.utexas.edu as technical report TR-PDS-1998-011.
6. C. Baquero and F. Moura. Causality in Autonomous Mobile Systems, 1999.
7. F. Torres-Rojas and M. Ahamad. Plausible Clocks: Constant Size Logical Clocks for Distributed Systems, 1996.
8. L. Bellissard, N. De Palma, A. Freyssinet, M. Herrmann, and S. Lacourte. *An Agent Platform for Reliable Asynchronous Distributed Programming.* Symposium on Reliable Distributed Systems, October 1999.
9. G. Agha, P. Wegner, and A. Yonezawa. *Research Directions in Concurrent Object-Oriented Systems.* MIT Press, Cambridge, 1993.
10. Jeff Magee, Jeff Kramer, and Morris Sloman. Constructing Distributed Systems in Conic. *IEEE Transactions on Software Engineering*, 15(6), 1989.
11. L. Bellissard, N. De Palma, D. Féliot, and M. Serrano. Abstract ADL. Technical report, C3DS Project First Year Report, 1999. available at `http://www.laas.research.ec.org/c3ds/trs/papers/13.pdf`.
12. S. Quaireau. Middleware Orienté Messages pour équipements autonomes. DEA report, June 2001.