Towards Zero-delay Recovery of Agents in Production Automation Systems

Eva Kühn, Richard Mordinyi, Mario Lang, and Adnan Selimovic
Complex Systems Design & Engineering Lab
Space-based Computing Group
Vienna University of Technology
{ek, rm, ml, as}@complang.tuwien.ac.at

Abstract

Multi-agent systems (MAS) is an accepted paradigm in safety-critical systems, like the production automation. Agents control the underlying machinery they are representing and interact with each other to achieve an optimal production rate. However, the key requirement is to minimize downtime of the production system in order to allow just-in-time delivery and minimal production costs. Nevertheless, as any other distributed system MAS are prone to failures as well, and consequently an agent may crash. In this case it is important that after recovery the agent achieves its optimal production state as fast as possible.

Given the recovery problem, in this paper we address the assimilation sub-problem, i.e. the problem of catching up with the external environment. We propose an abstraction framework called MozartSpaces that keeps the state information, necessary for an agent to behave correctly, up-to-date transparently to the agents. This allows the failed agent to continue its work after recovery with zero-delay in the state where it would have been without a crash.

1 Introduction

One of the main purposes of MAS in production automation domain is to satisfy business requirements [14], such as the optimal usage of the production system in order to allow just-in-time delivery and cause minimal production costs. However, as any other distributed system, MAS are prone to failures as well. Once a failed agent is up and running again, it is important that the time needed for the recovered agent to regain its "optimal" execution state for production is as minimal as possible. Since only one agent at a time is capable of controlling the underlying machinery, running several replicated agents of the same type concurrently is not possible, and thus a new instance has to be deployed and executed. However, the newly deployed agent is not aware of the circumstances in its new environment, requiring time consuming and extensive message exchange with neighboring agents to update its view on the system. Therefore, sophisticated recovery techniques have to be considered [17] [13]. However, a recovery from software faults is not sufficient enough, since the environment may have changed while a new instance of the agent was deployed. Therefore, the state information received at the time of recovery should already reflect occurred changes in the environment.

The assimilation problem [2] is a sub-problem of the agent recovery problem, and describes the issue of how an agent is capable of "catching up" with the external environment, by reaching a foregone considered state, in case the agent had not crashed. The goal of the presented work is to describe an abstraction framework, called MozartSpaces, that is used to store and distribute information needed by an agent to perform its objectives. Inherent components of MozartSpaces are containers [10] as shared data storages and aspects [9]. By separating computational logic and coordination information [8], the objectives of the agents are independently "stored" from the data needed to be capable of following its objectives. Aspects represent cross-cutting concerns and are used to distribute any information transparently on the agent's behalf to avoid data loss and to propagate changes in the environment by manipulating the shared container of neighboring agents [11]. Therefore, a failed agent's data is still accessible and already reflects changes in the environment at the time of recovery.

2 Related Work

2.1 Agent Recovery in MAS

Rollback-recovery protocols [5] are key strategies to achieve fault-tolerance and have been developed in order to increase reliability and availability of distributed systems. Those protocols can be classified into log-based and checkpoint-based protocols.

Log-based recovery [1] uses a message log for each agent, periodically recording its local state and log the mes-

978-0-7695-3801-3/09 $26.00 © 2009 IEEE
DOI 10.1109/WI-IAT.2009.170

sages it received after having recorded that state. Upon failure the state of the agent can be reincarnated [2] through the playback of logged messages. However, non-deterministic events have to be stored as well in order to ensure that the agents behavior is the same with respect to other agents. Additionally, it needs a lot of storage and processing for reincarnation. In case of coordinated checkpoints [12], a consistent set of checkpoints forms a recovery line so that all agents can roll back to a consistent global state.

However, in case of checkpoints it is difficult to roll back to a consistent state. Actions already performed by the underlying machines, like the assembly of product parts or painting, cannot be undone. On the other hand, log-based recovery does not store messages as long as the crashed agent is unavailable. This implies that the agent, once up and running again, is capable of restoring state information, but runs the risk of working with out-of-date information. This means that the agent is on the one hand capable of behaving correctly with respect to its objectives, but on the other hand it may use information that is in conflict with an overall "optimized" production system. Thus, the agent would have to analyse the environment any way to catch up and work within the requirements.

In [18] a so called dynamic recovery protocol has been introduced allowing the system to regain global consistency with low overhead. There, an agent buffers incoming operations during recovery. Then, the stored operations are replayed for further recovery. Although the approach may seem similar to the presented one, it differs in two ways. First, the protocol requires for each node an agent being responsible for recovery. In the presented approach the combined power of all production agents based on the shared memory concept facilitates recovery. Second, the protocol still requires a final synchronization step, whereas in the presented approach such actions are not necessary.

2.2 Space-based Computing

MozartSpaces is a Java implementation extending Linda [7]. It describs the usage of a logically shared memory, called *tuple space*. By means of the simple operations (`write`, `read`, `destructive read`, `eval`) it is a communication mechanism for parallel and distributed processes. In principal, the tuple space is a bag containing tuples with non-deterministic operation access. The main difference between traditional space implementations like [6] [15] [3] or [19] and MozartSpaces[1] is the container and aspect. It allows the storage of entries in a customizable structured and ordered way. The ordered, structured form of the space is achieved by specific customizable Coordinators [10], which are capable of distinguishing explicitly between data needed for coordination purposes and the payload itself. Aspects represent programming logic, are executed on the peer where the container is located and can be triggered before or after any operation is/was performed on the container. In contrast to aspects, MARS [3], TuCSon [4], and LIME [15] enable the modification of the operations' semantics by adding so called *reactions*. A reaction is defined by an instruction, which specifies the actions to be executed when a tuple matching a pattern is found in the tuple space. However, modification of the operation is meant in the sense of manipulating already existing tuples, but they cannot influence the execution of the operation itself.

2.3 Databases

Established database products like Oracle or DB2 are heavy-weight components, and as such the peers running the agents in the production automation system do not have the capacity to execute them. Alternatives would be light-weight databases, like Oracle Berkeley DB Java Edition[2], Apache Derby[3], hsqldb[4], H2[5], or db4o[6]. However, the drawback of databases is that they need a static data model of the entries they have to store, while containers allow the usage of several different Coordinators at the same time, enabling dynamic data models, and thus being schema-free. In case of db4o, accessing an entry is performed via query-by-example, like in tuple spaces [7]. However, in [10] it has been shown that containers allow an optimized realization of queries and coordination models. Finally, databases aim to store and retrieve long living data, while the scenario focuses on short lived data. Triggers can be seen as aspects as well, but are mainly meant for operations within the database itself.

3 Architecture

This section pictures the architecture of a zero-delay recovery concept with MozartSpaces in detail. It will describe the content of containers, the replication of data, the deployed aspects and the propagation of changes in the system.

3.1 Set up of the production system based on containers

A production system [14] consists of several different software agents each being responsible for an underlying

[1] to be downloaded at `http://www.mozartspaces.org`

[2] `http://www.oracle.com/database/berkeley-db/index.html`

[3] `http://db.apache.org/derby/`

[4] `http://hsqldb.org/`

[5] `http://www.h2database.com`

[6] `http://www.db4o.com/`

machinery. Such an agent may be: a **pallet agent (PA)** representing the transportation of a production part and knowing the next machine to be targeted by the real pallet, a **crossing agent (CA)** routing pallets towards the right direction according to a routing table, which may be structured according to specific optimization objectives, like pallet routing delay, a **conveyor belt agent (CBA)** transporting pallets from one crossing agent to another, a **machine agent (MA)** controlling robots for e.g., painting or assembling product parts, or a **strategy agent (SA)** which, based on the current usage rate of the production system, knows where to delegate pallets, so that by taking into consideration business requirements, a product is created in an efficient way.

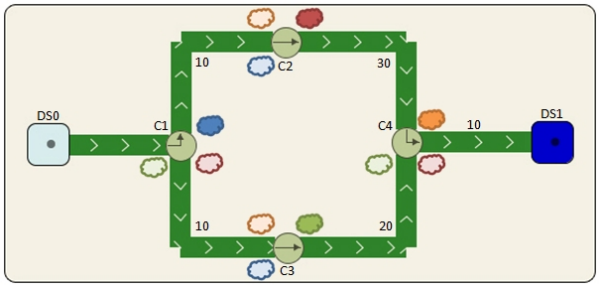

Figure 1. Simple production system

In this work crossing agents are of main importance. The idea is that each agent in the production system (figure 1) has a dedicated container for storing its data structures (like a routing table). When the system is started the very first time, all existing CAs exchange information about incoming and outgoing CBAs and the costs of each conveyor belt to get informed about the environment they are running in. This information is collected and stored in the container. For instance a container may store routes to a destination (MA) prioritized according to the costs of the conveyor belts. For instance, the ”cheapest” route from agent C1 to docking station DS1 controlled by an MA is reachable if the incoming pallet is forwarded to crossing agent C3 via the outgoing conveyor OC1.

3.2 Distributing containers

So far each container resides on the peer where its agent is executed. In case that peer fails, the agent's container is not accessible any more. This fact prevents other agents from updating the failed agents container and thus the failed agent is not informed about any changes in the environment. Therefore, containers have to be replicated and distributed.

In [9] we have described how to replicate containers transparently to the using application. There, it has been presented how to combine containers with an overlay network based on the Distributed Hast Table (DHT) concepts [16] to make such containers fault-tolerant. The name of the container is used as a key to retrieve its value, the actual URL to the container. The underlying DHT implementation already provides built-in replication mechanisms to distribute the key in the network. This mechanism is used to get informed about data movements due to failing peers and as such it triggers replication strategies installed as aspects on each container. By means of aspects it is possible to deploy several replication strategies to keep containers consistent, to increase fault-tolerance and their availability. What kind of replication strategy is deployed depends on business requirements and production optimization objectives.

Additionally, the DHT concept has been altered in a manner, which allows to store events with respect to locality. Mapping this approach to the production automation allows to store containers in different areas of the production system in order to even more increase availability of data, thus creating replication clusters (figures 2).

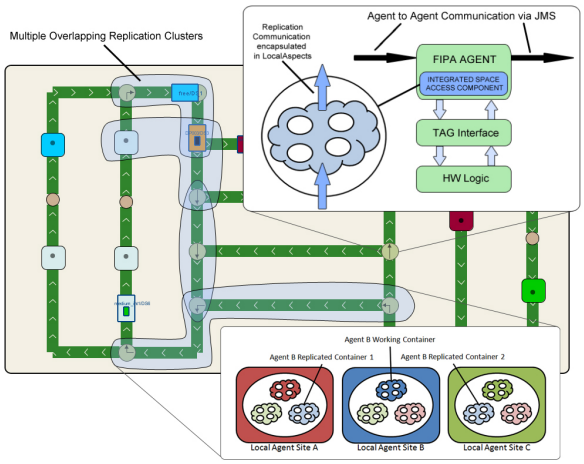

Figure 2. Agents with triple replicated containers using various DHT lookup areas (replication clusters)

Figure 2 shows three agents using their specific replication clusters consisting of three replicated containers.

3.3 Propagation of Changes

Consider a change in the described layout like the failure of crossing C3 in figure 1. The first (adjacent) component to discover the failure updates the components previous to the failing component - in our case the crossing C1. The routing information in said container is updated (the route over OC1 will be set to disabled by setting its costs to a negative value). Said update triggers a subsequent update

of the components previous to crossing C1 (encapsulated and implemented in local aspects of the containers). Further routing to DS1 will use the path over OC0.

This means that aspects deployed on containers analyze incoming events and based on the type of the event they manipulate the container of other agents transparently to those agents; thus creating a shared container. Since agent logic and state information is separated, with respect to agent failures, the state information of the failed agent can be updated as well. Even in the case, in which not just the agent but the entire host has crashed, there are several replicas available, and accessible via a DHT lookup for data updates. Therefore, when an agent recovers, it performs a lookup for its container, receiving one with already updated information about the environment.

4 Conclusion

In this paper we described a concept towards zero-delay recovery of agents in the production automation system based a Linda-like paradigm called MozartSpaces. The proposed concept allows reducing the complexity of agents since the efforts needed for coordinating with other agents has been reduced to two single operations. This allows the agent to work immediately in an "optimal" way with respect to the business requirement, namely overall optimal usage of the production system to allow just-in-time delivery and minimize production costs. Evaluations still need to verify the concept, but it seems that the usage of interconnected and replicated containers reduces the time needed to recover in case of multiple agent crashes. Therefore, future work will contain comparison with traditional JMS based communication and coordination protocols for replication and recovery in a distributed environment.

Acknowledgement: This work has been partially funded by the Complex Systems Design & Engineering Lab, Vienna University of Technology (`http://www.informatik.tuwien.ac.at/csde/`). The authors would also like to acknowledge the works of the Rockwell Automation Research Center, Czech Republic, in the field of the Manufacturing Agent Simulation Tool (MAST).

References

[1] L. Alvisi and K. Marzullo. Message logging: Pessimistic, optimistic, causal, and optimal. *IEEE Trans. Softw. Eng.*, 24(2):149–159, 1998.

[2] A. K. Bansal, K. Ramamohanarao, and A. S. Rao. Distributed storage of replicated beliefs to facilitate recovery of distributed intelligent agents. In *ATAL '97: Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages*, pages 77–91, London, UK, 1998. Springer-Verlag.

[3] G. Cabri, L. Leonardi, and F. Zambonelli. Mars: a programmable coordination architecture for mobile agents. *Internet Computing, IEEE*, 4(4):26–35, Jul/Aug 2000.

[4] M. Cremonini, A. Omicini, and F. Zambonelli. Coordination and access control in open distributed agent systems: The tucson approach, 2000.

[5] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.

[6] E. Freeman, K. Arnold, and S. Hupfer. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley Longman Ltd., Essex, UK, UK, 1999.

[7] D. Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[8] D. Gelernter and N. Carriero. Coordination languages and their significance. *Commun. ACM*, 35(2):97–107, 1992.

[9] E. Kühn, R. Mordinyi, H.-D. Goiss, S. Bessler, and S. Tomic. A p2p network of space containers for efficient management of spatial-temporal data in intelligent transportation scenarios. *Accepted for the International Symposium on Parallel and Distributed Computing, ISPDC (TechRep at http://tinyurl.com/c796j4)*, 2009.

[10] E. Kühn, R. Mordinyi, L. Keszthelyi, and C. Schreiber. Introducing the concept of customizable structured spaces for agent coordination in the production automation domain. *The 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[11] E. Kühn, R. Mordinyi, L. Keszthelyi, and C. Schreiber. Towards efficient publish/subscribe scenarios in intelligent transportation systems with aspect-oriented space containers. *Accepted for 8th Working IEEE/IFIP Conference on Software Architecture (WICSA'09)*, 2009.

[12] P.-J. Leu and B. K. Bhargava. Concurrent robust checkpointing and recovery in distributed systems. In *Proceedings of the Fourth International Conference on Data Engineering*, pages 154–163, Washington, DC, USA, 1988. IEEE Computer Society.

[13] H. F. Li, Z. Wei, and D. Goswami. Quasi-atomic recovery for distributed agents. *Parallel Comput.*, 32(10):733–758, 2006.

[14] M. Merdan, T. Moser, D. Wahyudin, S. Biffl, and P. Vrba. Simulation of workflow scheduling strategies using the mast test management system. *10th International Conference on Control, Automation, Robotics and Vision*, 2008.

[15] A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Trans. Softw. Eng. Methodol.*, 15(3):279–328, 2006.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Proceedings of the 18th IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware 2001)*, pages 329–350, 2001.

[17] R. Strom and S. Yemini. Optimistic recovery in distributed systems. *ACM Trans. Comput. Syst.*, 3(3):204–226, 1985.

[18] Y. Wang and X. Liu. Agent based dynamic recovery protocol in distributed databases. pages 274–280, Oct. 2003.

[19] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. T spaces. *IBM Systems Journal*, 37(3):454–474, 1998.