Modelling Fault-Tolerant Systems using BDI Agents

Felipe Rech Meneguzzi

Pontifícia Universidade Católica do Rio Grande do Sul, PPGCC
Av. Ipiranga 6681, CEP 90619-900 Porto Alegre, Brasil
felipe@cpts.pucrs.br

Abstract. A number of mechanisms for providing fault tolerance in distributed systems have been proposed for the traditional paradigms of system development. As Agent-Oriented Programming (AOP) becomes a more attractive model for distributed system construction, new possibilities of designing more flexible dependable systems are foreseen. Therefore, fault-tolerant mechanisms should be defined so that dependable multi-agent systems might be developed. In this paper we describe a way of implementing a fault-tolerant multi-agent system, which is inspired in some of the mechanisms proposed for distributed systems.
Key words: Multi-Agent Systems, Fault Tolerance

1 Introduction

Various mechanisms and techniques have been proposed in order to provide dependability for distributed systems developed using what now are standard programming techniques like Object Orientation (OO) [10, 13]. Meanwhile, Agent Oriented Programming (AOP) [4] is being proposed as a new approach to the development of distributed systems. While a consensus on the specifics of AOP is far from being achieved, some authors have already expressed their concerns on the development of fault-tolerant mechanisms for Multi-Agent Systems (MAS) [5, 3]. While MAS are more flexible than traditional distributed systems due to an agent's ability to reason about unpredicted situations and more reliable due to the fact that agents are easily replicated, failures in the system itself and whatever it controls are still possible. Therefore, it is important for a MAS to have some kind of mechanism that allows a group of agents performing some task to try to bring the system to a stable state in case of a failure, or otherwise notify the whole system that a catastrophic failure has occurred.

In this paper we propose a way to implement agent-oriented fault tolerance based on the Beliefs, Desires and Intentions (BDI) model of agents [1]. In particular we base our proposal on a specific type of BDI agent, namely an improved version of the X-BDI agent model [6]. Our proposal is inspired on two mechanisms of fault-tolerance for distributed systems, namely Coordinated Atomic Actions [12] and Dependable Multiparty Interactions (DMI) [13]. Our purpose in defining such a mechanism is twofold, to provide the means for a MAS to be fault-tolerant and to add the flexibility of BDI agents to the development of dependable systems. This paper is divided into three main parts: Section 2 describes the works upon which this paper is based; in Section 3 we describe a way to implement fault tolerance in a MAS. Finally, in Section 4 we discuss the work presented in this paper.

2 Related Work

This section briefly describes the work that has a direct relationship with the topics proposed in Section 3. We start by describing the BDI model and its components in Section 2.1, as well as the specific BDI model used to underpin our proposal in Section 2.2. Finally, we describe the main mechanism which inspired this work in Section 2.3.

2.1 BDI Agents

In the context of multi-agent systems research, the BDI model is one of most widely known and studied models of deliberative agents. It is characterized by the use of Beliefs, Desires and Intentions as abstractions for the description of a system's behaviour and was originated by a philosophical model of human practical reasoning, later developed into a computational theory [9]. The components that characterize the BDI model can be briefly described as follows [7]:

– **Beliefs** represent the agent's expectancy regarding the current world state or the possibility that a given course of action will lead to a given world state;
– **Desires** represent a set of preferences the agent has regarding a set of world states (this set is not necessarily consistent);
– **Intentions** represent the agent's commitment regarding a given course of action, constraining the consideration of new objectives.

2.2 The X-BDI Agent Model

The X-BDI agent model was created to allow a formal agent specification to be directly executed [6]. That is accomplished through the use of a formalism called Extended Logic Programming with explicit negation (ELP). An X-BDI agent is described by three main components:

– A set of actions that essentially specify the abilities an agent has. An action is comprised of a set of pre-conditions that states when it is possible for the agent to execute the action and a set of effects that decribe the result of the execution of that action;
– A set of desires that specify a set of possible goals the agent might try to accomplish. A desire is comprised of a pre-condition that states when that goal becomes relevant to the agent. It also has a priority value used by the agent to resolve which goals to pursue when multiple goals become relevant but are not consistent;

– A set of initial beliefs used to initialize the knowledge of the agent in a particular domain.

An agent receives input about the state of the world from its sensors. This information is used by the agent to revise its beliefs so that they are both consistent and reflect the state of the environment as perceived by the agent. The deliberation process of an agent starts by selecting a set of relevant desires called *Elligible Desires* comprised of all the desires whose pre-condition have been satisfied. These desires are then used by the agent to construct various sets of consistent desires ordered by their priority values. These sets are sent to an external planning algorithm[1], and the higher priority set for which a plan can be generated comprise the *Candidate Desires*. Candidate Desires become *Primary Intentions* that represent the commitment of the agent to those goals. The steps of the plan generated to fulfill the Candidate Desires become the agent's *Relative Intentions*, representing the agent's commitment to that particular course of action. The approach used in X-BDI to perform runtime planning instead of using the standard plan library gives X-BDI a greater flexibility in problem solving than previous BDI agent architectures.

2.3 The DMI Fault-Tolerant Mechanism

Several diferent types of systems often involve complex concurrent activities. In some cases, these activities may be working together, i.e. *cooperating*, to solve a given problem; in other cases they can be completely *independent* needing to *compete* for shared common resources. In practice, orthogonal types of concurrency may happen at the same time in complex systems, which will thus require a supporting mechanism to cope with faults that might happen during agents interactions.

In this paper, we use a fault-tolerant mechanism to provide means for handling concurrent exceptions and synchronisation upon exit of several participants (we can understand participants as objects, agents, processes, threads, ...) that come together to execute a cooperating activity. After the cooperating activity is finished, all participants leave the interaction and continue their normal (possibly independent) execution. This mechanism is called *Dependable Multiparty Interaction* (DMI).

Basically a DMI has the following properties:

– Synchronisation upon entry of the interaction participants;
– Checking of a pre-condition before the participants start to execute their activity in the interaction, hence the need for having synchronization upon entry;
– Checking of an assertion after the interaction has finished to guarantee that a set of post-conditions have been satisfied by interaction participants;

[1] The current implementation of X-BDI uses Graphplan as the planning algorithm

– Atomicity of external data to guarantee that wrong, intermediate, information is not passed to participants of other interactions before the related interaction has finished;
– Each participant executes a different part of the interaction, i.e. the interaction activities is split among the participants;
– The number of participants is variable in each different interaction.

3 An Agent-Based Fault Tolerant System

In this section we describe how a MAS might implement fault tolerance in the process of carrying out some task required by the system. In this paper we do not address the issues regarding dependability within the agent system itself, though there are other works that address these issues [3, 2], rather we describe a "communication protocol" among multiple agents that allows them to deal with failures occuring during an agent's interaction with the environment in which they operate, or among themselves. All activities in the interaction respect the DMI properties mentioned in Section 2.3.

Other authors have already described fault-tolerance mechanisms for agent systems [5]. These works are based on BDI agents whose deliberation process uses a plan-library in order to determine the actions an agent might execute in order to achieve a given goal. In this paper we base our approach in a type of agent that determines the sequence of actions that are necessary to achieve a goal at runtime through the use of a planning algorithm (see Section 2.2), thus providing a greater level of flexibility.

Our approach to fault tolerance in MAS is similar to the architecture of *One Manager Object and Several Role Objects* described in [11], we shall call our architecture *Dependable Agent Interaction (DAI)*. In our approach, a group of agents "team up" to fulfill some task designated by a Master Agent, receive information regarding the task at hand, perform the task while trying to deal with faults that might emerge during the execution of the task, and then leave the group. It is important to point out that we are not trying to deal with the issue of coalition formation in our proposal [8], though this mechanism might be proven usefull as a protocol for the dynamic creation of DAIs.

Therefore, we define DAI as being a group of Agents whose components fulfill two basic responsibilities: one Master and various Participant. The Master Agent controls the entry and exit of Participant Agents in the interaction as well as the synchronization among the agents. The Master Agent functions as a proxy for the agents involved in the interaction, thus trying to encapsulate the current state of processing. Participant Agents receive tasks from the Master Agent, try to acomplish these tasks and deal with any faults that might occur, informing the Master Agent when they fail to do so. Although the conceptual use of one Master Agent may lead to the belief that the system is especially sensitive to the single point of failure limitations, the seamless replication of Middle Agents proposed by [2] is a possible solution to the problem. The "protocol" of a DAI can be summarized as follows:

- The Master Agent distributes tasks to as many agents as he has tasks for as well as information associated with the fulfillment of these tasks;
- These tasks are actually properties or parts of a world-state the Participant Agent has to achieve, these tasks are represented in the Participant Agent as a Desire with a priority value higher than the priorities of all other desires, except for fault recovery ones;
- When enough agents have joined the interaction and all the tasks have been distributed, the Master Agent notifies every Participant Agent that he can start to carry out their task;
- Upon notification, the involved agents start their planning process so that they can find a course of action to fulfill their appointed tasks. At this point two things might happen: i) all Participant Agents successfully generate a plan to fulfill their tasks; ii) one or more agents fail to generate a plan, in which case the Master Agent is notified that one of its tasks is not possible. The Master Agent then notifies all Participants that a failure has occurred and starts a global recovery task;
- If all the Participants generate a plan, they start carrying out the plans they generated. During plan execution one of three things can happen: i) the Agents complete their tasks successfully; ii) a fault occurs during plan execution by an agent, but the agent successfully replans and finishes its task; iii) A fault occurs during plan execution and the agent fails to deal with it. In that case it will inform the Master Agent who will start a global recovery task;
- If an agent achieves its task successfully, it notifies the Master Agent;
- When every agent has notified the Master of its success, the Master informs whoever requested the DAI to be executed of its success;
- When a Participant fails to recover from a local error, it informs the Master of a global failure;
- When the Master agent detects a global failure either by receiving notification from an agent or by detecting that some agent has stopped responding, it notifies all Participants that an error has occurred and asks them to perform some global recovery task, the desires related to the fulfillment of a recovery task always have top priority;
- If in the process of global recovery another global failure occurs, a catastrophic failure has occurred and the Master informs whoever requested the DAI that the mechanism itself has failed.

4 Concluding Remarks

This paper has presented some initial ideas of how fault-tolerant MAS systems can be modelled. In order to fulfill fault tolerance properties, agents' interactions emulate a special multiparty interaction mechanism called *Dependable Multiparty Interaction* (DMI). The DMI provides the agents involved in the interaction with the capability of handling concurrent exceptions that might happen during the interaction. The main contribution of the paper is to bring fault tolerance requirements into the design of multiple BDI agents, thus merging the flexibility of dynamic planning BDI agents with a fault-tolerant mechanism. The work proposed in this paper is part of a greater project involving the development of BDI agents and one of our main goals in this work is to define a fault tolerance mechanism for BDI agents.

Acknowledgments. We would like to thank Professors Michael Móra and Lucia Giraffa that help in the discussion of the ideas on agents presented in this paper. Avelino Zorzo, which supervises this work, is supported by CNPq/Brazil. Felipe Meneguzzi is supported by HP Brazil.

References

1. Wiebe Van der Hoek and Michael Wooldridge. Towards a logic of rational agency. *Logic Journal of the IGPL*, 11(2):133–157, March 2003.
2. Alan Fedoruk and Ralph Deters. Using dynamic proxy agent replicate groups to improve fault-tolerance in multi-agent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 990–991. ACM Press, 2003.
3. Z. Guessoum, J. P. Briot, S. Charpentier, O. Marin, and P. Sens. A fault-tolerant multi-agent framework. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 672–673. ACM Press, 2002.
4. Nicholas R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.
5. Sanjeev Kumar and Philip R. Cohen. Towards a fault-tolerant multi-agent system architecture. In *Proceedings of the fourth international conference on Autonomous agents*, pages 459–466. ACM Press, 2000.
6. Michael C. Móra. *A Formal and Executable Model of BDI Agents.* PhD thesis, CPGCC/UFRGS, 1999. In Portuguese.
7. Jörg P. Müller. *The Design of Intelligent Agents: A Layered Approach.* Springer-Verlag, Germany, 1996.
8. Mark Sims, Claudia V. Goldman, and Victor Lesser. Self-organization through bottom-up coalition formation. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 867–874. ACM Press, 2003.
9. Michael Wooldridge. *Reasoning about Rational Agents.* The MIT Press, 2000.
10. Jie Xu, Brian Randell, Alexander Romanovsky, Cecilia Rubira, Robert J. Stroud, and Zie Wu. Fault tolerance in concurrent object-oriented software through coordinated error recovery. In *25th International Symposium on Fault-Tolerant Computing*, pages 450–457. IEEE Computer Society Press, 1995.
11. Avelino F. Zorzo. A production cell controlled by dependable multiparty interactions. Technical Report 667, University of Newcastle upon Tyne, Newcastle upon Tyne, UK, March 1999.
12. Avelino F. Zorzo, Alexander B. Romanovsky, Jie Xu, Brian Randell, Robert J. Stroud, and Ian Welch. Using coordinated atomic actions to design dependable distributed object systems. *Software - Practice and Experience*, 29(8):677–697, January 1999.
13. Avelino F. Zorzo and Robert J. Stroud. A distributed object-oriented framework for dependable multiparty interactions. In *Proceedings of the 1999 ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 435–446. ACM Press, 1999.