Evaluating Multiagent Systems: A Record/Replay Approach

Marc-Philippe Huget
Leibniz-IMAG/MAGMA
46, Avenue Félix Viallet
F-38031 Grenoble Cedex
France
Marc-Philippe.Huget@imag.fr

Yves Demazeau
Leibniz-IMAG/MAGMA
46, Avenue Félix Viallet
F-38031 Grenoble Cedex
France
Yves.Demazeau@imag.fr

Abstract

The adoption of multiagent systems by companies and users requires that it is possible to verify the properties of the multiagent systems. The main candidate of this verification is model checking but as sketched in the paper, it presents drawbacks such as combinatorial explosion or it does not take the environment into account. Model checking seems to be restricted to small systems with few agents. In this paper, we present a different approach used in testing called the Record/Replay mechanism. We add as well a post-mortem analysis. This paper describes the advantages of this approach and how we have implemented it in the MadKit platform.

1. Introduction

Agents gain acceptance as an interesting technology in domains where autonomy and user representation are at stake, but users are still cautious employing them. One reason among others is the absence of trust in this new technology: there is no way to verify that either agents and multiagent systems integrate user requirements, behave correctly or are not malicious. One possible way to fix this problem is to provide to users (and designers), algorithms and tools for evaluating multiagent systems.

To the best of our knowledge, there is no current work that proposes to evaluate an industrial application of multiagent systems and for several reasons:

1. Increased complexity since there are several distributed processes that run autonomously and concurrently (the scaling problem);
2. Amount of data since systems can be up to thousands of agents, each coming with data (user profiles, knowledge, beliefs, etc.).

Irreproducibility effect is certainly the main problem regarding evaluation. It means that it is not ensured that two executions of the system will lead to the same state. As a consequence, looking for a particular error can be difficult if it is not impossible to reproduce it each time. This is due to the nondeterministic characteristics of multiagent systems at the environment level or at the decision level in agents. We will show in Section 2 how we can deal—to a certain extent—with this problem.

There exist at least three approaches for evaluating multiagent systems: (1) theorem proving, (2) model checking, and (3) testing. Theorem proving comes from logic systems and corresponds to check that a set of formulas satisfies a goal via inferences. Model checking builds a model of a system and verifies that a temporal logic formula holds for the model [2]. Finally, testing consists in applying test cases to the system and checks if it behaves properly. In this paper, we focus on a specific kind of testing called the Record/Replay mechanism [4] used in regression testing. The Record/Replay mechanism is a test performed during the execution of the system either in simulation or in production. It is realized through system inspection. The record phase records actions in the system (memory, environment, data update, messages, etc.). When an error occurs in the system, designers have a system that they can play and replay until they found the error and fixed it. Contrarily to model checking, testing checks that agents behave properly rather than agents are proved correct. In our approach, the Record/Replay mechanism is coupled to testing via post-mortem analysis. It uses the events and data stored during the record phase and checks properties without reexecuting the system.

Section 2 describes the Record/Replay mechanism and its application is given in Section 3. Section 3 describes as well evaluation via post-mortem analysis and the testing platform based on MadKit [3]. Section 4 evaluates the Record/Replay mechanism approach and gives future directions.
2. The Record/Replay Mechanism

When programs are nondeterministic (partly due to their distribution on the network), it can be difficult to check them since two runs of the same system do not necessarily end in the same state since processes can receive non-deterministic events and data. They can decide to change their way to do tasks. For this reason, classic testing method such as black box testing and white box testing can be inefficient. Agents are autonomous entities that act independently. As a consequence, they can change their behaviors in two successive runs since their intentions have changed. To cope the nondeterminism of multiagent systems, we propose to use a method used for debugging parallel systems called the Record/Replay mechanism [4]. The Record/Replay mechanism is decomposed into two phases: a first phase that traces the system execution, particularly non-deterministic events, hence called the record phase and a second phase where this information is reused to guide a deterministic reexecution, hence called the replay phase. This is a deterministic replay since we constrain the system to behave as it did in the record phase.

The information stored depends on what designers want to replay. Actually, there exist two kinds of execution replay: content-based and ordering-based execution replay. The content-based execution replay considers storing all the instructions from systems (instructions from programs as well as memory). In this case, it is possible to replay exactly the faulty system. This approach has low level of abstraction and consumes I/O bandwidth to store data in trace files. Moreover, using content-based execution replay can affect the system to debug since it consumes resources to store information (the probe effect). The ordering-based execution replay only stores the non-deterministic events in order to make sure they will be replayed in the same order. As stated, the ordering-based execution replay has high level of abstraction. In case of ordering-based execution replay, the record phase uses vector clocks to store the event ordering. Hybrid approaches mixing content-based and ordering-based execution replays offer the best of the two approaches. This is the one, we will follow in this paper.

3. Testing Multiagent Systems with Record/Replay

This section describes how the Record/Replay mechanism is performed on multiagent systems, what kind of information the system stores during the record phase and the platform used. We add a second kind of testing via post-mortem analysis where data recorded during the record phase are inspected.

3.1. The Record Phase

As stated in Section 2, the aim of the record phase is to record data and events that need to be stored in the context of replay (and post-mortem analysis). Usually, only non-deterministic data and events need to be recorded since there are the only ones that are difficult to reproduce, particularly if the system has to reach a specific system state. Since we also consider a post-mortem analysis, the record phase is not limited to non-deterministic data and events but to all events and data that are required for the post-mortem analysis. Actually, we instantiate the Record/Replay mechanism to the specific multiagent system domain by tracing some specific features:

- Non-deterministic data and data used during the post-mortem analysis. If designers want to inspect a specific variable for a post-mortem analysis, the record phase does that via a primitive called trace. For instance, \texttt{trace("a", a);} indicates the variable \texttt{a} is traced at that moment and the value is stored. The function has to be called each time the variable \texttt{a} is modified. Non-deterministic data can be user inputs or Web data for instance.

The selection of variables to trace depends if we are interested in replaying or in post-mortem analysis. If the only interest is the replay phase, then it is just necessary to record non-deterministic data since other data do not change at runtime. If designers want to perform a post-mortem analysis, it is important to select variables to analyze. The selection of variables is of designer’s concern since they are able to find which variables can be of interest. Designers have to be cautious not to record all the variables due to the probe effect that can affect the execution.

- Messages since it is possible that agents do not use the same messages each time. Messages are traced via the group message tracer provided by the MadKit platform (see Section 3.2). It reduces the intrusiveness since there is nothing to do within agents to record messages.

- Agents belong to groups. The record phase in our approach considers recording the group evolution: new agents entering the group, agents leaving the group. It is then possible to check some social properties like norms. Once again, there is no intrusion within agents to record these data since the platform has an organization tracer provided by the MadKit platform that stores the different groups and the agent membership (see Section 3.2).

- The environment is non-deterministic. We make the assumption that it is possible without too much computation to retrieve an exact picture of the environment at any moment of execution. We use for that a global
structure shared by all agents in which agents and resources position is recorded. Time is defined discrete, as a consequence, recording the environment corresponds to recording modification at each time unit. The recording is performed by the environment tracer agent added to the MadKit platform.

Traces are stored in files, one per agent and one per tracer (organization, group message and environment). Each record is preceded by a timestamp to help the post-mortem analysis.

### 3.2. The Testing Platform

As stated in Section 3.1, the record phase is performed through some monitors either described within the agents or available within the platform. We propose to take benefit of current agent platforms to evaluate multiagent systems. We opt for the MadKit platform [3] since the platform has already two tracer agents: the group message tracer and the organization tracer. We use these two agents even if they are superseded by the group observer agent in order to ease separation of concern.

The group message tracer agent is responsible to trace messages sent within the group. The message recording is composed of several fields: the sender, the receiver, the message class, the content and the date. The message class defines how the message is described. We modify the group message tracer agent in order to take account of the record/replay phase. As a consequence, each time the group message tracer agent receives a message, it records it in a file. During the replay phase, the group message tracer agent reorders the messages based on the trace file. To this purpose, the group message tracer agent keeps the messages as long as the first message (based on the trace file) is not arrived. It then sends the next messages if it has them. To this purpose, agents do not send directly messages to agents but to the group message tracer.

The organization tracer agent is responsible to record group and role actions: when agents enter or leave a group, or when they add or remove a role. The organization tracer agent records several pieces of information: the involved agent, the action, the group, the role and the date. Actions are one of these values: add_member_role, remove_member_role, found_group, leave_group and kill_agent. We modify the organization tracer agent to take account of the record phase. It is then possible to record group and role actions in a trace file. The organization tracer agent is not used in the replay phase.

We add a third tracer agent to these tracer agents: the environment tracer agent. The environment tracer agent is responsible to record the environment initial configuration and modification. To this purpose, the environment tracer agent checks the environment at the end of each time interval and records what the modifications (new agent position, resource production or resource consumption) are. The modifications are stored in a trace file.

Agents executed on the platform do not need special attention except the use of the trace and the getfromTrace primitives that are used respectively in the record phase and in the replay phase. Agents, in the replay phase, send messages to the group message tracer and not directly to recipients in order to repeat the exact order.

### 3.3. The Replay Phase

The replay phase uses the data recorded during the record phase to replay a specific agent or the multiagent systems. The content-based execution replay is performed as follows:

1. The initial environment is extracted from the trace of the environment tracer agent. It means that resources and agents are located at the position they were at the beginning of the record phase.
2. Each time, agents need a non-deterministic variable data, they get it from the trace file. For instance, here is an excerpt of the code of an agent:

   ```java
   if (MODE_RECORD) { /*record*/
       input = getUserInput();
       trace("input", input);
   } else { /*replay*/
       input = getfromTrace("input");
   }
   ```

   In the context of ordering-based execution replay, it is necessary that messages arrived in the same order to agents. Agents are constrained in the way they have to send the message to the group message tracer and not to the recipient in order to ensure message ordering. The group message tracer is responsible to reorder messages based on the trace file and deliver them to agents in this order.

   We do not use, for the moment, the environment tracer in the ordering-based execution replay. We plan to use the environment trace as checkpoints: the user specifies which environment situation she wants to consider as initial situation, then the replay is performed from this environment situation. The agents are necessarily moved to a situation compatible with this environment situation, that is, agents have the state they have just at the moment of this environment situation.

   Due to space restriction, we do not speak about reexecuting a single agent. It is possible in the Record/Replay mechanism to execute a specific process and let it evolving and believing it really interacts with other processes. This is realized thanks to the record phase that can simulate the messages of other processes.
3.4. Post-mortem Analysis

The post-mortem analysis uses the data recorded during the record phase. It is thus crucial to define the relevant set of variables to record during the record phase. This post-mortem analysis is currently a manual task realized by the user. The first step is to define the property to verify and the variables, she needs. Then, after the record phase, the designer extracts the data from the trace files and verifies the properties. For instance, if the designers wants to check that after receiving a FIPA inform message, the recipient believes the content of the message. The designer extracts the message from the group message tracer file and notes the timestamp of the message. She has now to check if there is a belief about the content of the message dated after the message receipt. If yes, the recipient believes the content.

Future work on the post-mortem analysis will consider automating this process. For instance, an analysis agent will receive a temporal logic formula and instruments agents to record data. After the execution of the multiagent system, the analysis agent will extract data from trace files and verifies the properties.

4. Conclusion and Perspectives

Even if model checking is a powerful tool for the validation of (a)synchronous systems, this approach presents limitations in the context of multiagent system validation such as combinatorial explosion, absence of local variables or environment configuration. As a consequence, model checking multiagent systems seems to be only adopted on small (or toy) problem with a limited number of agents and with small computation skills. This is particularly the case in MABLE [5]. In this paper, we propose the use of the Record/Replay mechanism. This mechanism is used in debugging parallel or distributed programs. Its main advantage is to deal with scaling and combinatorial explosion since only non-deterministic events (and optionally data) are recorded. We instantiate the Record/Replay mechanism to evaluate multiagent systems via several modifications: (1) the record phase records group structures, the environment configuration; and deterministic data and events if they are at stake during the post-mortem analysis, (2) the replay phase is more intrusive since it requires to enforce agents to use data recorded during the record phase.

We demonstrate this Record/Replay mechanism on the MadKit platform by adding one agent (environment tracer) to record the environment configuration and modifying the two provided tracers (group message and organization) in order to record the trace and replay the message ordering. Agents that need to be traced contain some supplementary code and particularly, the trace and the getfromTrace primitives to, respectively, record data and replay data.

Besides the Record/Replay mechanism, a post-mortem analysis is added. This phase considers the data recorded during the record phase and does some verification of properties. It is then possible to record data and verifies the relation between them. This could be of help, for instance, when verifying that agents are compliant to the ACL semantics or when verifying that an agent respects some norms if it belongs to a specific group. It is interesting to note that this approach of the record phase and post-mortem analysis is also considered in runtime verification of software [1].

Several directions of work are already considered:

1. Providing a tool that inserts the trace primitive where required. The idea is to offer to designers the ability to enter a logic formula. A tool will find the related variables based on the logic formula.
2. Providing a visual editor for the record phase thus designers can see data evolution.
3. Doing some performance validation on this approach. In order to fully justify the Record/Replay mechanism, we will apply the same multiagent systems to both model checking and our approach. We then evaluate the performance of these two approaches and how they react to scaling up.
4. Doing complexity analysis in order to evaluate the overhead of the record phase during the execution of the multiagent system.

References