Pre-Proceedings of 8th International Workshop on Engineering Multi-Agent Systems
(EMAS 2020 @ AAMAS), 8-9 May 2020, Auckland, New Zealand.
Eds. Cristina Baroglio, Jomi F. Hübner, and Michael Winikoff.

Exploiting Simulation for MAS Programming and Engineering—The JaCaMo-sim Platform

Alessandro Ricci[1], Angelo Croatti[1],
Rafael H. Bordini[2], Jomi F. Hübner[3], and Olivier Boissier[4]

[1] DISI, University of Bologna, Italy, a.ricci|a.croatti@unibo.it
[2] POLI-PUCRS, Brazil, rafael.bordini@pucrs.br
[3] Federal University of Santa Catarina, Brazil, jomi.hubner@ufsc.br
[4] Univ. Lyon, MINES Saint-Etienne, CNRS Lab., France, olivier.boissier@emse.fr

Abstract. Simulation can be an important conceptual and practical tool to support the engineering of multi-agent systems (MAS), in different ways —testing in particular. In this paper we consider the case in which simulation is applied and exploited directly upon a MAS developed using an existing agent programming/-MAS programming platform. That is: without requiring to model and simulate agents and their environment using a different platform (e.g., an agent-based simulation one). In particular, we describe the design and prototype implementation of JaCaMo-sim, an extension of the JaCaMo platform that makes it possible to both run and simulate the execution of MAS programs based on BDI agents written in Jason, situated in artifact-based environments developed in CArtAgO. The tool can be useful for different aspects that concern MAS engineering, from MAS testing at development time to – in perspective – agent decision making support at runtime.

1 Introduction

An important feature for developers when programming agents or multi-agent systems is the possibility to run the system in test-bed environments, before deploying it. This could be useful to evaluate performance as well as to address issues before they become problems, so as to improve system reliability, or to check the effectiveness of how agents and the whole MAS has been programmed. In some cases, this would account for simply programming *simulated environments*, reproducing the interested situations and scenarios that the MAS is going to deal with, after deployment. In some other cases, this accounts for analysing the temporal evolution of the agent and MAS behaviour, eventually considering different kinds of execution environments.

In the literature, the role that simulation can play for Agent-Oriented Software Engineering has been already remarked and explored in several directions [21,20]. A main one is about *testing* [16]. Testing of agents and multi-agent systems is typically harder than conventional software systems, due to aspects such as autonomy, interaction, concurrency, distribution [11]. Exploiting simulation for testing an agent-based system in general accounts for creating *models* and performing experiments with those models to answer questions about the system. Models could concern the environment where

agents are supposed to run (creating virtual environments), as well as agents themselves, reproducing agents' behaviour at some level of abstraction.

In this context, as pointed out by Uhrmacher [21], an important direction and challenge for simulation in AOSE is about having hybrid development/execution/simulation environments that would allow to execute agents *as they are* and to switch arbitrarily between the execution in the real environment and test environment. At the same time, this hybrid development/execution/simulation environments should allow agents to be an integral part of the experimental setting and as such perceivable and controllable. That is: having tools supporting a graceful transformation from *simulation* to *emulation* [21]. This challenge was suggested in the literature two decades ago and is still an open issue.

In our research we aim at exploring and tacking this challenge in the context of agent-oriented programming and multi-agent oriented programming, in particular using the JaCaMo platform [3]. This platform allows for programming MAS integrating different and independent programming dimensions: agents are programmed using the Jason BDI agent programming language; the environment can be programmed using the CArtAgO framework, based on the A&A (agents and artifacts) conceptual model; and organisation can be specified and programmed using the Moise framework. In this paper we present and discuss the model, design and first prototype of a platform that makes it possible to run a JaCaMo program both in real mode and simulation mode. When executing in simulation mode, the program is executed like a simulation by simulators, that is abstracting from the real physical hardware and environment where the MAS is executed, using simulated time. MAS execution in this case becomes a time-controlled simulation, based on the DES (Discrete Event Simulation) model [6]. Differently from existing simulation tools in MAS, in our case the *model* to be simulated is the MAS program itself.

We believe that the idea and the tool could be useful for three main aspects that concern MAS engineering. The first obvious one is about testing. The tool would allow to test/observe a MAS behaviour in any conditions without being bound to the availability of specific execution/deployment environments. For instance, with this tool, a complex distributed MAS may be run in a simulated mode on a single computer, where all aspects about user interaction, external environment, etc. are simulated. Like for every simulation/simulator, when running in simulated-time, the result of the simulation is independent from the computer or system used to run the simulation. Just, when using faster computers, the simulation is going to take less time. The second one – still related to the development time – is about supporting the development of specific application contexts where agent/MAS development may need to run agents/MAS in simulated environments. For instance, when agents need to be *trained* before being deployed, like in the case of Reinforcement Learning. The third case is about runtime, where the tool could be used by agents to help their decision making, in particular by predicting the effect of their actions by running a simulation from the current state of the MAS.

The paper is organised as follows. After an account of related works (Section 2), we describe the idea behind the JaCaMo-sim platform, modeling MAS execution as a Discrete Event Simulation (Section 3), Then, we describe how the model has been implemented, i.e. the architecture of JaCaMo-sim platform (Section 4) and an example

showing the tool at work (Section 5). We conclude the paper by sketching the road ahead that we see for this research line (Section 6).

2 Related Works

Our work is related first of all to existing approaches in the literature that explored the value and use of simulation for AOSE [21,20]. These include proposals that use simulation to support the process of MAS development, such in the case of PASSIM [4]; to support the agent-based design and testing [17]; to define integrated approaches for the development and validation of MAS [7]; or, to engineer self-organising MAS [8]. Platforms such as Multi-Agent System Simulator (MASS) [22] and Sensible Agents [1] have been designed to investigate the performance of agents and multi-agent systems in complex environments.

Our contribution is especially related to those applying simulation to testing of agents and MAS – which is a main challenge in MAS engineering [11,12] – and, in particular, with those that explore the integration of BDI agents and agent programming languages/purpose with simulation environments. In the literature, research works exploring such an integration have been proposed more in the context of agent-based simulation (ABS) and multi-agent based systems (MABS). A main example is [19], which describes a three-tiered BDI-ABM architecture, to integrate existing simulation platforms (e.g., MATSim) and the BDI frameworks (e.g., JACK) as independent and uncoupled parts that interact by means of an action/perception interface, and using a time-step based approach to advance the execution. Further more recent examples include [10,5]. Our approach has a different scope, being not targeted to ABS/MABS but AOSE. In spite of the different perspective, from a technical point of view in our case a Discrete Event Simulation model is adopted and the BDI Platform itself (JaCaMo in our case) is extended so as to support a simulation execution modality.

Finally, a research work that we see strongly related to the proposal in this paper is the Brahms framework [18]. Brahms has been primarily developed as modelling and simulation environment for work practices, but finally it has been used also as agent-based platform to develop real systems. The same platform can be used for both simulating and running a MAS system. The path taken in our paper has been somewhat in the opposite direction: we started from a platform used for developing and running MAS and we extended it in order to support the simulation of the MAS – keeping the "agents as they are" [21], without needing their modelling in a different language.

3 The Approach

In this section we describe how the idea works at a model level, abstracting from implementation details. The idea is based on two main points. The first one is that, at the model level, the execution of a (JaCaMo) MAS, at the bottom level, is *event-driven*. The agent part (Jason) is based on a BDI reasoning cycle [13], which can be modelled as a well-defined sequence of events concerning the sense, deliberate and act stages. The communication part is managed by the Jason side – possibly exploiting different ACL platforms, like Jade [2] – and can be modelled in terms of events as well. The environment

part (CArtAgO) can be described too in terms of events [15], in which computations proceed as soon as a new operation is requested on artifacts (corresponding to action on the agent side). In order to have observable effects, the execution of operations updates artifact's observable state, generating events. The implementation of the organisation part (MOISE) is based on agents and artifacts [9].

Being event-driven, it is straightforward to model the execution of a MAS program as a Discrete Event Simulation (DES) [6]. In particular, MAS program execution is modelled as a system in which state changes can be represented by a collection of discrete events, occurring at a certain time. In DES, a state change implies that an event occurs and the states of entities remain constant between events. In MAS program execution we can assume this, by choosing a proper level of abstraction about states and events, so as to abstract from changes that are not considered relevant. When executing a MAS program, events are scheduled and executed by the control flows used by the execution platform (that can be distributed). On the agent side, there are control flows used to move on agent reasoning cycles and on environment side there are control flows used to execute operations on artifacts. To execute a MAS in a *simulated mode*, event scheduling and execution are intercepted, so as to be governed by a classic DES simulation loop [6], deciding which events should be scheduled next, according to the time planned for them. A next-event approach to advancing time can be used, i.e. after all state changes have been made at the time corresponding to a particular event, simulated time is advanced to the time of the next event and the event is executed. Differently from simple DES, in our case, being MAS generally a distributed system, we do not want to assume a single simulated timeline Every agent and artifact have their own independent timeline, we cannot assume a priori a common unique timeline.

The second main point is about the *external* environment, how it is modelled and how the MAS interacts with it. In JaCaMo, any aspects that concern the external environment of a MAS are meant to be modelled/encapsulated into artifacts, These specific artifacts are also called *boundary artifacts*, since they are the boundary inside the MAS that enables/mediates the interaction with the external environment. Examples range from artifacts representing GUI (enabling the interaction with human users) to external devices (e.g., a printer, a sensor) and services (e.g., Internet-based API for maps). In a simulated execution, boundary artifacts are replaced with a version that implements the simulated behaviour, however preserving the interface in terms of operations (actions) and observable state (percepts).

3.1 Execution contexts, events and activities

We introduce here a set of concepts to capture the idea in spite of the specific agent platform/model adopted. We introduce the concept of Execution Context (EC) to model any locus of activity of the MAS, equipped with its own timeline. In our case, we have an EC for each agent, referred as agent EC, and for all basic abstractions of the MAS which have an independent existence and timeline with respect to agents. These elements include the environment and the communication medium used to enable agent communications. In the case of JaCaMo, the environment is modelled in terms of artifacts and workspaces, so an EC is introduced for each artifact (artifact EC) and workspace (workspace EC). An EC is introduced also for the communication medium (referred

as comm EC), enabling speech-act based message passing among agents. Each EC is characterised by its own clock *Ts* to keep track of the (simulated) time.

The dynamics/behaviour of each EC is described in terms of events occurring there. Events have no duration, they occur in a precise time of the EC timeline. Each event is characterised by a timestamp *ts*, assigned using the clock Ts, representing when the event happened (or is scheduled to happen) inside the EC. It is worth remarking here the difference between event generation and even execution. Event generation concerns when the event is created and scheduled. Event execution concerns when the scheduled event actually happens, occurs. In normal MAS execution, event generation and execution almost coincide. When executing in simulated mode instead, events are scheduled to happen in the future.

Events occurring in the same EC are totally ordered. Instead, events of different EC can only be partially ordered, exploiting a *causal relationship* between them. For instance, the action request done by an agent – that appears in the agent EC – is the root of a chain of causally-related events that result in starting the execution of the corresponding operation by the artifact – that appears in the artifact EC. The same holds for agent communication, involving the sending of a message and the receipt of the same message, two events that are part of the same causal chain across two different ECs.

The concept of *activity* is introduced to represent something relevant occurring between two related events—that correspond to the beginning of the activity and the end of the activity. For instance: the beginning of a reasoning cycle and the end of the same reasoning cycle. For activities, a notion of duration can be defined, as the difference between the timestamps of the two events, being them part of the same EC. Activities can overlap or can be wrapped by other activities. For instance: a sense activity, marking the sense stage inside the reasoning cycle, is wrapped/included in the reasoning cycle activity.

Some activities may span over multiple ECs. A main example concerns agent communication, in which the beginning event is the *send* action executed in the act stage of the sender agent and the end event is the message receipt occurring in the sense stage by the receiver agent. Another main example concerns the execution of an external action, as an activity whose beginning event (the action request) and the end event (the perception about action completion or failure) occurs in the same EC (the agent EC), however involving a chain of events that occurs both in the EC of the artifact hosting the operation and the workspace hosting the artifact.

The duration of activities in these cases *cannot* be computed simply as the difference of the timestamps of the beginning and end events, because these two events or events in the causal chain belong to different ECs, possibly with independent clocks. To tackle this problem, we introduce a notion of *synchronisation events* that bind together two different ECs in chains of causally-related events. For instance, the event representing an action request on an agent EC and the notification of the same action to be dispatched on the workspace EC that hosts the artifact. In this case, the execution of the first event in one EC causes the synchronous execution of the second event in the other EC, where the meaning of *synchronous* depends on the model of time defined for specific simulation by the developer/modeller.

A core set of events and activities We identified a first core set of events modelling the event-driven execution of a MAS program. These events can be split in three main categories:

- events concerning agent reasoning cycle execution, involving only the EC of a single agent;
- events concerning agent communication, involving the EC of two agents and of the communication medium; and
- events concerning agent environment interaction, involving the EC of an agent, an artifact and the workspace hosting the artifact.

Table 1 (reported in the appendix) shows a partial list of events concerning the agent EC related to the agent reasoning cycle, including relevant activities connecting the events. Figure 1 (on the right) shows events and activities on a timeline. The partial list includes, in particular, events occurring in the sense stage, since they are related also to events that concern agent communication and interaction with the environment. Examples of events not included in the list are events occurring in the deliberation stage – such as the creation of a new goal or the instantiation of a new intention – and events occurring in the act stage, such as the execution of internal actions.

Events and activities related to agent communication are listed in Table 2 (in appendix) and shown on a time line in Figure 1 (on the left). Agent communication involves three different ECs: the EC of the sender agent, of the receiver agent and of the communication medium exploited to deliver the message.

Finally, the full list of events and activities concerning agent-environment interaction is reported in Table 3 (events) and Table 4 (activities)—both in the appendix. The the timeline is shown in Figure 2. Agent-environment interaction involves three different ECs as well: the EC of the agent executing an external action or observing some artifact; the EC of the artifact providing the operation to be executed or the observable state to be observed; and the EC of the workspace hosting the artifact and joined by the agent, functioning as a glue.

The agent-environment interaction concerns two scenarios, shown in Figure 2. The first concerns an agent requesting an action, which triggers the execution of an operation hosted by an artifact. When (if) the operation execution completes (or fails), an action event is generated and notified on the agent side. The second concerns the generation of an observable event on the environment side – that could concern either the update of observable properties of an artifact, or the generation of a signal – notified as a percept on the agent side. It is worth remarking that in CArtAgO, on which JaCaMo is based, even predefined actions (e.g., to create an artifact, to lookup artifact, and so forth) are modelled as actions provided by some existing artifact [14]. In this paper we do not consider events and activities that concern interaction between artifacts—that is, artifact executing operation over another artifact (called linked operation in CArtAgO).

3.2 The Simulation Loop

The execution of the simulation follows a classic event-scheduling approach as found in DES, adapted so that the MAS execution platform (JaCaMo in our case) is used to

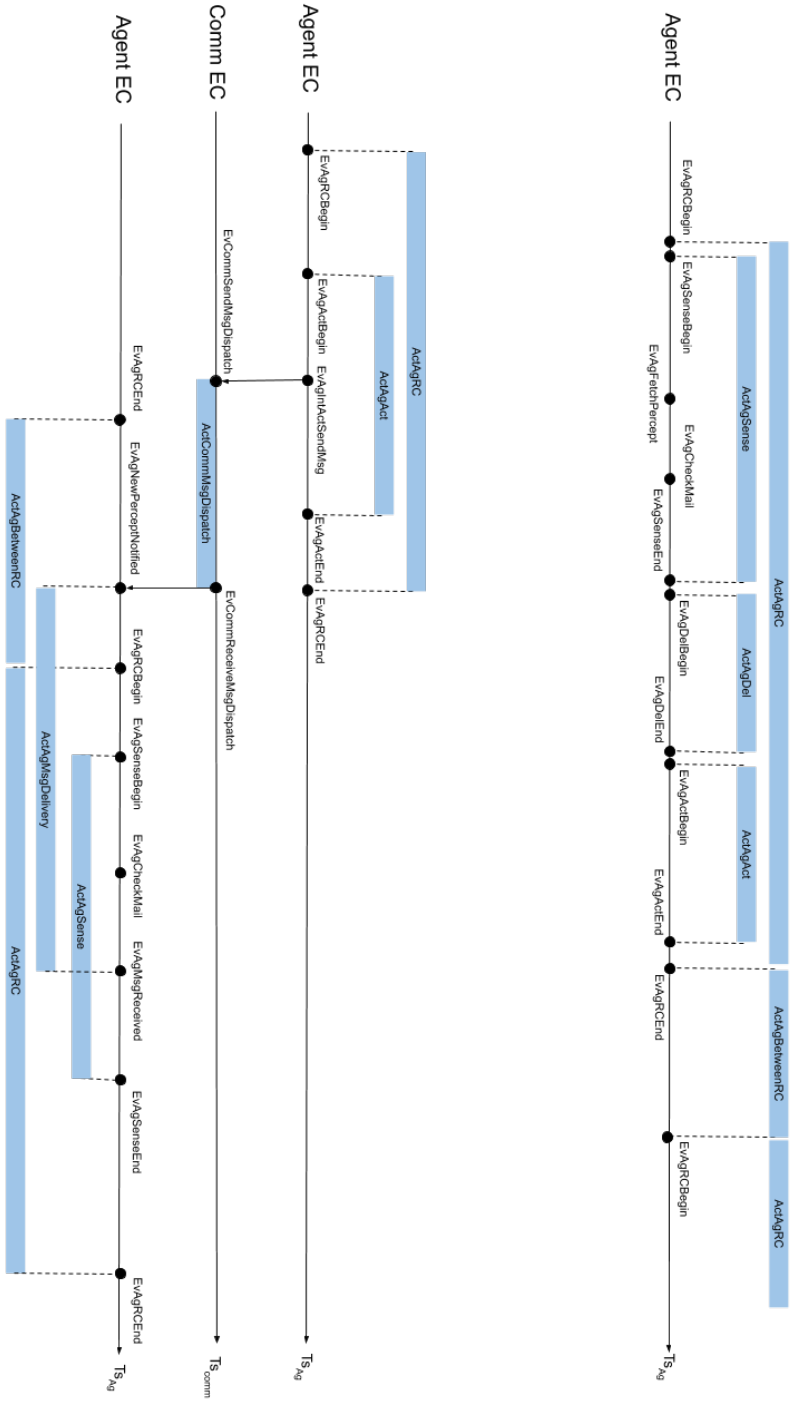

Fig. 1: Diagrams representing events and activities related to communication between agents, involving also the communication media EC.

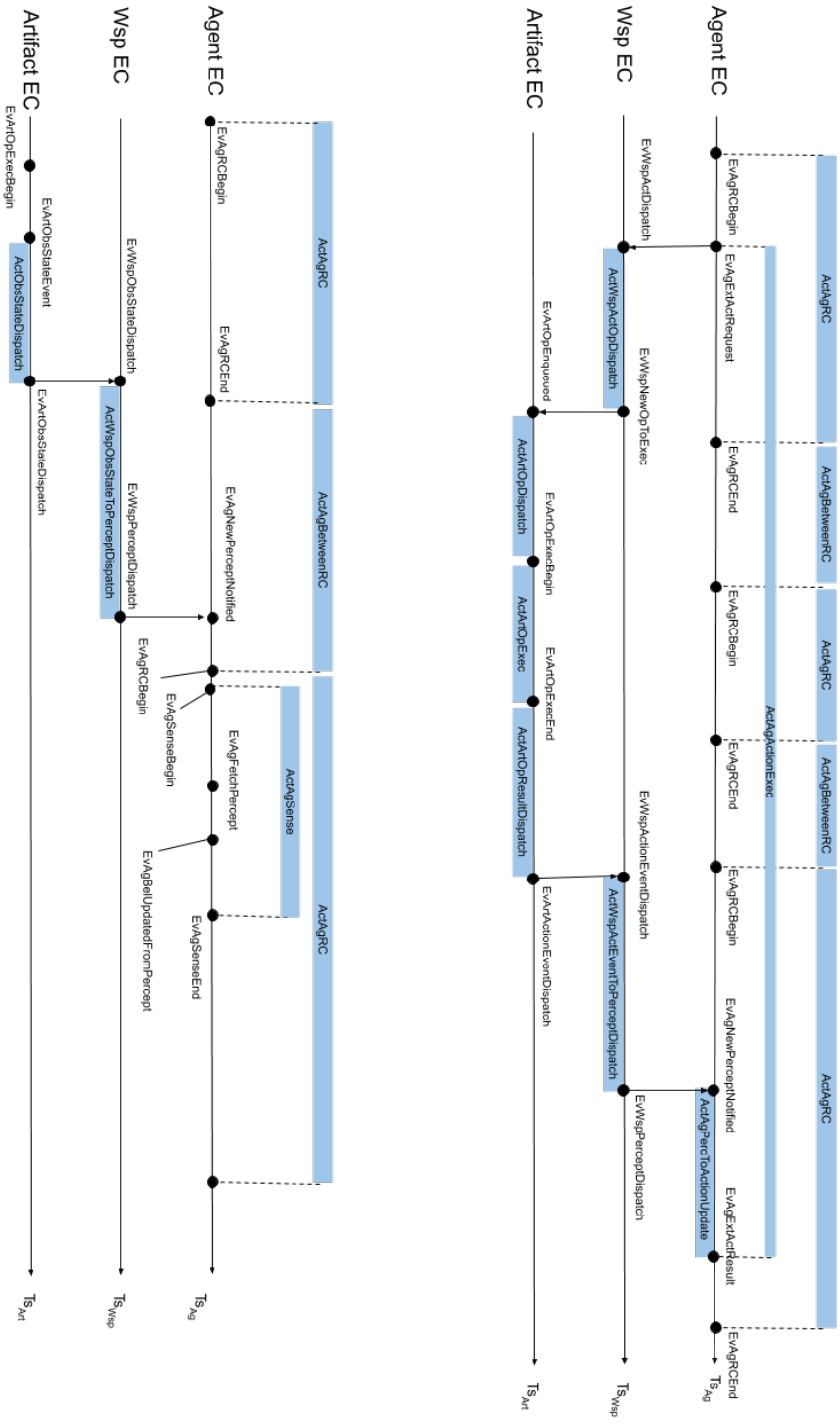

Fig. 2: Diagrams representing events and activities related to agent EC and artifact EC.

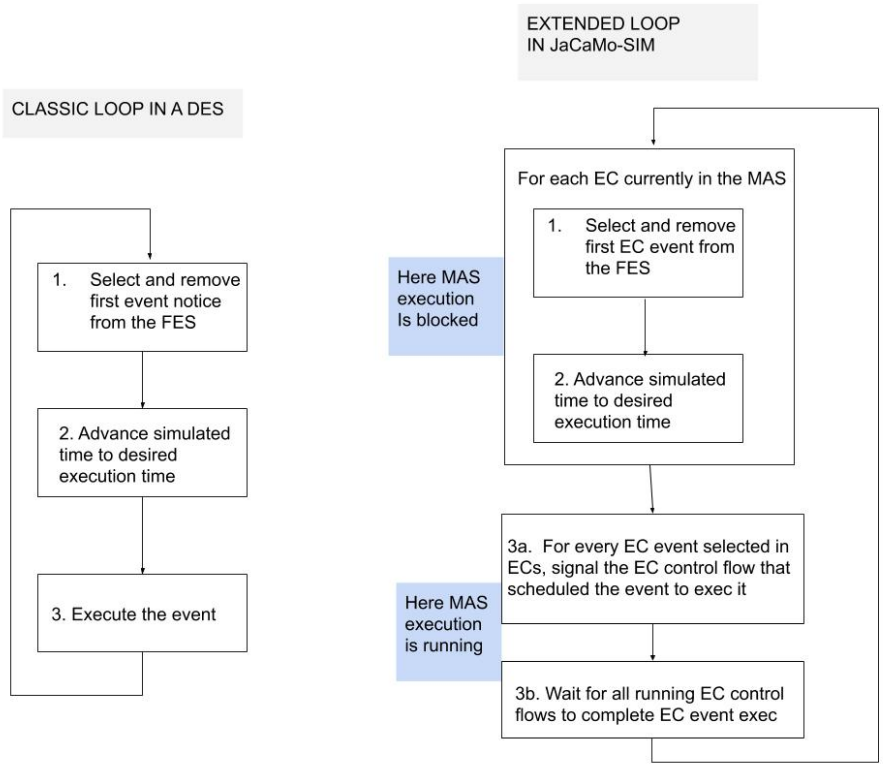

Fig. 3: The simulation loop – on the left: the classic DES version; on the right: the one adopted in JaCaMo-sim

run the model (the MAS program). Each EC has its own *FES* (Future Event Set), which represents the set of events that have been scheduled to be executed (i.e. to occur) in the future [6]. Given the event-driven behaviour, the dynamics/execution of each EC can be modelled/tracked as a state that atomically evolves given the execution (occurrence) of events. The execution of an event causes the generation of events that are scheduled in the FES of the EC of the event and possibly in other ECs. Each event is decorated with its timestamp ts computed by a *time assignment function* establishing when the event is going to happen in the future.

Like in the DES case, the simulator behaviour is given by a *loop*, run by its own control flow (depicted in Figure 3). Differently from the DES case, the execution of events is carried by the JaCaMo platforms (concurrently), by means of the EC control flows (that are part of the Jason and CArtAgO scheduler systems). Basically there are 2 kind of EC control flows: the ones running the agent reasoning cycles and the ones running operation executions on artifacts in workspaces. In executing events, EC control

flows may schedule EC events in different ECs. At each iteration, the simulator loop considers the (dynamic) set of all ECs currently in execution and, before choosing next event to be scheduled, the loop must be sure that all running control flows completed current event execution. This is a synchronisation point between the MAS execution platform and the simulation loop. So at each iteration the simulation loop: *(1)* waits all control flows to be blocked (sync point); *(2)* selects the next event to be scheduled for an EC; and *(3)* unblocks the corresponding control flow on the platform side which is waiting to execute the event.

3.3 The Time Assignment Function

A key aspect of the simulation is the assignment of the (simulated) time when scheduling a new event, that is: specifying when it is going to happen in the future. This time could be any value greater or equal than the current time of the EC where the event occurs. If the event represents the end of some activity, then the event time is equal to the current time of the EC plus the duration that the activity is supposed to have. This time could be either random or not, could either depend on the specific state of the EC or not.

Actually, the specific strategy adopted to assign a time to events is application specific. Therefore the simulator is meant to provide maximum flexibility to developers/modellers for defining that function, to allow for recreating the specific situation to be tested/experimented. Such a flexibility includes also the possibility to implement different time/distribution models in defining the strategy—from a centralized case, where all agents and the environment are supposed to run in the same node, sharing a common clock, to fully decentralized and distributed cases, where agents and artifacts are running on different nodes, and the e.g. Internet is used as underlying network for enabling both agent communication and agent-environment interaction.

4 First Implementation

The JaCaMo-sim platform[5] provides a first implementation of the approach described before. The platform is a lightweight extension of the basic JaCaMo platform. A new component called (JCM) Execution Controller is added. This component is called by the main platform each time a new event – which is relevant, given the model discussed before – is going to be scheduled or executed. The calls are performed by the control flows that are used inside the JaCaMo platform (Jason side, CArtAgO side) to execute agents and artifacts.

The Execution Controller component could be configured to work in different modalities: (1) *normal mode*; (2) *tracking mode*; (3) *simulation mode* (see Figure 4). In normal mode, the Execution Controller is almost an empty component, not creating any overhead over MAS execution. In *tracking mode*, the Execution Controller is a lightweight layer just tracking time of events in ECs. Some components (Viewer, Logger) are used to visualize/log tracked data. The clock of the ECs in this case is directly the clock of the machine(s) running the EC, so events are decorated with timestamps that are directly

[5] Available here: `https://github.com/jacamo-lang/jacamo-sim`

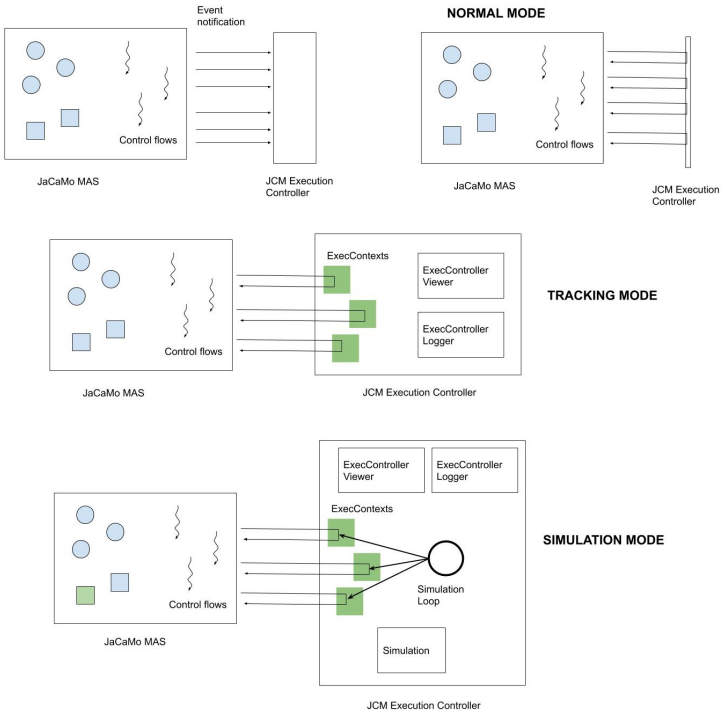

Fig. 4: JaCaMo-sim Platform.

the wall time. This modality is useful for profiling purposes, in particular to check the duration of activities.

In *simulation mode* the Execution Controller contains the simulator loop described in previous section, controlling event scheduling and execution. In current API, to configure the parameters and temporal behaviour of the simulation, a `Simulation` base class is provided – to be extended – and a concrete instance of its extensions can be used to initialize the platform in simulation mode. The interface of the `Simulation` class includes a method to be overridden (called `assignTime`) implementing the Time Assignment Function described in previous section, that is: the method is called each time an event is scheduled, so as to decorate it with a specific time. Besides this method, the `Simulation` API includes also methods to directly specify the duration of activities, if available. Some concrete examples are shown in next section.

5 The Tool at Work

The JaCaMo-sim distribution includes some simple examples that can be used to play with the tool, involving agent-environment interaction and agent communication. Example 1 is a very simple case about agent-environment interaction, involving a `tester_agent` and a `Counter` artifact. The artifact provides an `inc` operation and a

```
!main_test.

+!main_test
 <- println("start");
    makeArtifact("counter", "ex1.Counter", [], Id);
    focus(Id);
    inc.

+count(V) <- println("count value: ",V).
```

Fig. 5: Source code of the `tester_agent`, in example 1 (available in the repository).

```
class CentralizedSimulation extends Simulation {
  private Random gen;
  private long time;

  public CentralizedSimulation() {
    gen = new Random(1);
    time = 0;

    setActivityDuration("ActAgRC", (ECActivity act, ExecContext ctx) -> {
      return 1000 + gen.nextInt(1000); });

    setActivityDuration("ActArtOpExec", (ECActivity act, ExecContext ctx) -> {
      if (ctx.getId().equals("counter")) {
        return 5000 + gen.nextInt(1000);
      } else return 0;
    });}

   public void assignTime(ECEvent ev, ExecContext ctx) {
     long time = ctx.getCurrentTimeInMicroSec();
     if (ev.getName().equals("EvAgExtActResult")) {

       ECEvent evActReq = ev.getEventInTheCausalChain("EvAgExtActRequest");
       ECActivity act = ev.getActivityInTheCausalChain("ActArtOpExec");

       if (evActReq != null && act != null) {
         long dur = act.getDurationInMicroSec();
         long startedTime = evActReq.getTimeInMicroSec();
         long evt = startedTime + dur;
         ev.setTime(evt/1000, evt);
       }}
     ...}
   ...}

/* main spawning the simulation */

public class RunTest {
  public static void main(String args[]) throws Exception {
    ExecutionController contr = ExecutionController.getExecController();
    Simulation sim = new CentralizedSimulation();
    contr.initSimulationMode(sim);  /* initialize the execution in simulation mode */
    // contr.initTrackingMode();

    /* spawn the MAS */
    jason.infra.centralised.RunCentralisedMAS.main(
      new String[] { "src/test/jcmsim/example-1/main.mas2j" });
  }
}
```

Fig. 6: Source code of the main, configuring and launching the simulation.

`value` observable property, which is update each time the operation is executed. The agent simply creates an instance of the artifact, called `counter`, start observing it and executes an `inc` action. When the agent perceives a change on the `count` observable property, it prints a current value on the console. Figure 5 shows the source code of the Jason agent and Figure 6 a snippet of the Java application, configuring and launching the example.

The simulation is configured so that: the duration of the ActAgRC activity (representing the reasoning cycle activity) is a random value between 1 and 2 ms; the duration of the `inc` operation execution is a random value between 5 and 6 ms; all the other activities (not specified) are meant to have a zero duration. A centralised model of time is adopted (this depends on the implementation of the `assignTime` method overridden from the `Simulation` class).

Figure 7 instead shows an excerpt of the output produced launching the example 1, in tracking mode (on left) and in simulation mode (on the right). It shows a portion of the events and activities concerning the `tester_agent` and the `counter` artifact, with in evidence the timings (in microseconds) related to events and activities concerning the reasoning cycle, the execution of actions and operations, the events related to the `count` observable property changes and corresponding percepts. The timings in the simulated mode, in particular the duration of reasoning cycles and of operation execution, corresponds to the time assignment function configured in the simulation objects.

The other examples available in the distribution makes it possible to test the tool in the case of agent communication (example 2) and agent-environment with boundary artifacts (example 3) —in particular a GUI artifact, enabling the interaction with human user.

6 The Road Ahead

We consider this paper just a first step of a broad research direction that aims at exploring the integration of simulation and agent/MAS programming. On the one hand, the idea presented in this paper should be general enough to be applied to any agent/MAS programming platform, besides JaCaMo, for exploring any interesting scenarios related to that integration. On the other hand, current tool – which is based the specific JaCaMo platform – provides a very basic support to that purpose, suffering of limitations that will be tackled in future work.

The main directions that we see for the road ahead for this research line include, first of all, a deeper analysis and understanding of the specific use of simulation for MAS programming, eventually using some reference case studies to give more concreteness to the exploration. As mentioned in the first part of the paper, such a use is not limited to testing or validation, but may include also the use of the simulation at runtime to support agent decision making, to reason about the effect of actions. Another main direction is about defining a formalisation of the tool, capturing main concepts beyond the specific platform used, eventually exploring the use of existing formal frameworks such as DEVS [23], and related tools. A further interesting direction concerns the organization dimension, which has not been considered in this paper, that is: introducing a set of events and activities that concerns the organisation level and exploiting simulations to

```
AGENT: tester_agent - start time: 1582896117914

Events:
[0] [event: reasoning cycle begin | num-cycle: 1 ]
[1] [event: fetch percept | num-cycle: 1 ]
[6] [event: check mail | num-cycle: 1 ]
[8] [event: ext action req
       | makeArtifact("counter","ex1.Counter",[],Id)]
[11] [event: reasoning cycle end | num-cycle: 1 ]
[13] [event: new action event notified
       | action id: 1 | makeArtifact succeeded ]
[14] [event: reasoning cycle begin | num-cycle: 2 ]
...
[20] [event: ext action req | inc]
[20] [event: reasoning cycle end | num-cycle: 4 ]
[20] [event: new obs state notified
       | percept id: 10 | counter ]
[20] [event: reasoning cycle begin | num-cycle: 5 ]
[20] [event: fetch percept | num-cycle: 5 ]
[21] [event: new action event notified
       | action id: 3 | inc succeeded ]
[21] [event: BB updated with obs state event
       | percept 10 from counter]

...

Activities:
[0] [activity: reasoning cycle | num-cycle: 1 ]
     duration: 11 ms ( 10713 us )
[8] [activity: action exec | act-id: 1
     | makeArtifact | succeeded ]
     duration: 6 ms ( 6209 us )
[11] [activity: between reasoning cycles
     | num-cycles: 1 - 2 ]  duration: 3 ms ( 3117 us )
...
[20] [activity: action exec | act-id: 3
    | inc | succeeded ]  duration: 1 ms ( 1783 us )
...
```

```
AGENT: tester_agent - start time: 0

Events:
[0] [event: reasoning cycle begin | num-cycle: 1 ]
[0] [event: fetch percept | num-cycle: 1 ]
[0] [event: check mail | num-cycle: 1 ]
[0] [event: ext action req
    | makeArtifact("counter","ex1.Counter",[],Id)]
[1] [event: reasoning cycle end | num-cycle: 1 ]
[1] [event: reasoning cycle begin | num-cycle: 2 ]
...
[7] [event: ext action req | inc]
[9] [event: reasoning cycle end | num-cycle: 6 ]
...
[11] [event: reasoning cycle begin | num-cycle: 8 ]
[11] [event: fetch percept | num-cycle: 8 ]
[11] [event: new obs state notified
    | percept id: 10 | counter ]
...
[13] [event: reasoning cycle end | num-cycle: 8 ]
[13] [event: reasoning cycle begin | num-cycle: 9 ]
[13] [event: fetch percept | num-cycle: 9 ]
[13] [event: BB updated with obs state event
    | percept 10 from counter]
...
[13] [event: new action event notified
    | action id: 3 | inc succeeded ]
...
[15] [event: fetch percept | num-cycle: 10 ]
[15] [event: ext action result | act-id: 3 | success ]

Activities:
[0] [activity: reasoning cycle | num-cycle: 1 ]
    duration: 1 ms ( 1985 us )
[0] [activity: action exec | act-id: 1
    | makeArtifact | succeeded ]
    duration: 3 ms ( 3573 us )
[1] [activity: between reasoning cycles
    | num-cycles: 1 - 2 ]  duration: 0 ms ( 0 us )
...
[7] [activity: action exec | act-id: 3
    | inc | succeeded ]  duration: 8 ms ( 7064 us )
...
```

```
-----------------------------------
ARTIFACT: counter - start time: 1582896117934

Events:
[0] [event: art new op to dispatch | act-id: 3
    | inc on counter by tester_agent]
[0] [event: op exec begin | act-id: 3
    | inc on counter by tester_agent ]
[0] [event: art new obs state event | counter]
[1] [event: op exec end | act-id: 3
    | inc on counter ]
[1] [event: action event dispatch | act-id: 3 | inc ]
...
Activities:
[0] [activity: op dispatched | act-id: 3
    | inc on counter by tester_agent]
    duration: 0 ms ( 62 us )
[0] [activity: op execution | act-id: 3
    | inc on counter by tester_agent]
    duration: 1 ms ( 931 us )
[1] [activity: op result dispatch
    | act-id: 3 | inc]  duration: 0 ms ( 17 us )
...
```

```
-----------------------------------
ARTIFACT: counter - start time: 7

Events:
[0] [event: art new op to dispatch | act-id: 3
   | inc on counter by tester_agent]
[0] [event: op exec begin | act-id: 3
    | inc on counter by tester_agent ]
[0] [event: art new obs state event | counter]
[6] [event: op exec end | act-id: 3
    | inc on counter ]
[6] [event: action event dispatch | act-id: 3 | inc ]
...

Activities:
[0] [activity: op dispatched | act-id: 3
    | inc on counter by tester_agent]
    duration: 0 ms ( 0 us )
[0] [activity: op execution | act-id: 3
    | inc on counter by tester_agent]
    duration: 6 ms ( 5606 us )
[6] [activity: op result dispatch | act-id: 3
    | inc]  duration: 0 ms ( 0 us )
...
```

Fig. 7: Output logged by the tool running the MAS of example 1 both in tracking mode (on the left) and in simulation mode (on the right).

analyse the behaviour of the system at that level of abstraction. A generalisation of this point is about making the tool such extensible so that it would make it possible to define a new layer of events and activities – on top of the core set ones – corresponding to some (high) level of abstraction to the MAS and MAS behaviour, typically more near to the domain level.

References

1. Barber, K.S., McKay, R., MacMahon, M., Martin, C.E., Lam, D.N., Goel, A., Han, D.C., Kim, J.: Sensible agents: An implemented multi-agent system and testbed. In: Proceedings of the Fifth International Conference on Autonomous Agents. p. 92âĂŞ99. AGENTS âĂŹ01, Association for Computing Machinery, New York, NY, USA (2001). https://doi.org/10.1145/375735.376007, https://doi.org/10.1145/375735.376007
2. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley Series in Agent Technology, John Wiley & Sons (2007)
3. Boissier, O., Bordini, R., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. Science of Computer Programming **78**(6), 747–761 (2013). https://doi.org/10.1016/j.scico.2011.10.004
4. Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., Russo, W.: Passim: A simulation-based process for the development of multi-agent systems. Int. J. Agent-Oriented Softw. Eng. **2**(2), 132âĂŞ170 (Feb 2008). https://doi.org/10.1504/IJAOSE.2008.017313, https://doi.org/10.1504/IJAOSE.2008.017313
5. Davoust, A., Gavigan, P., Ruiz-Martin, C., Guillermo, Trabes, Esfandiari, B., Wainer, G.A., James, J.J.: An architecture for integrating bdi agents with a simulation environment. In: EMAS 2019 (2019)
6. Fishman, G.: Discrete-event Simulation: Modeling, Programming, and Analysis. Springer (2001)
7. Fortino, G., Garro, A., Russo, W.: An integrated approach for the development and validation of multi-agent systems. Comput. Syst. Sci. Eng. **20** (07 2005)
8. Gardelli, L., Viroli, M., Omicini, A.: On the role of simulations in engineering self-organising mas: The case of an intrusion detection system in tucson. In: Brueckner, S.A., Di Marzo Serugendo, G., Hales, D., Zambonelli, F. (eds.) Engineering Self-Organising Systems. pp. 153–166. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
9. Hübner, J.F., Boissier, O., Kitio, R., Ricci, A.: Instrumenting multi-agent organisations with organisational artifacts and agents: "Giving the organisational power back to the agents". Journal of Autonomous Agents and Multi-Agent Systems **20**(3), 369–400 (2010). https://doi.org/10.1007/s10458-009-9084-y
10. Larsen, J.B.: Going beyond bdi for agent-based simulation. Journal of Information and Telecommunication **3**(4), 446–464 (2019)
11. Miles, S., Winikoff, M., Cranefield, S., Nguyen, C., Perini, A., Tonella, P., Harman, M., Luck, M.: Why testing autonomous agents is hard and what can be done about it. URL http://www. pa. icar. cnr. it/cossentino/AOSETF10/docs/miles. pdf. AOSE Technical Forum (01 2010)
12. Nguyen, C.D., Perini, A., Bernon, C., Pavón, J., Thangarajah, J.: Testing in multi-agent systems. In: Gleizes, M.P., Gomez-Sanz, J.J. (eds.) Agent-Oriented Software Engineering X. pp. 180–190. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
13. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, January 22-25, 1996, Proceedings. LNCS, vol. 1038, pp. 42–55. Springer (1996)

14. Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: an artifact-based perspective. Autonomous Agents and Multi-Agent Systems **23**(2), 158–192 (Sep 2011)
15. Ricci, A., Viroli, M., Piunti, M.: Formalising the environment in mas programming: A formal model for artifact-based environments. In: Braubach, L., Briot, J.P., Thangarajah, J. (eds.) Programming Multi-Agent Systems. pp. 133–150. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
16. Röhl, M., Uhrmacher, A.M.: Controlled experimentation with agents — models and implementations. In: Gleizes, M.P., Omicini, A., Zambonelli, F. (eds.) Engineering Societies in the Agents World V. pp. 292–304. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
17. Sarjoughian, H., Zeigler, B., Hall, S.: A layered modeling and simulation architecture for agent-based system development. Proceedings of the IEEE **89**, 201 – 213 (03 2001). https://doi.org/10.1109/5.910855
18. Sierhuis, M., Hoof, R.: Brahms: A multi-agent modelling environment for simulating work processes and practices. International Journal of Simulation and Process Modelling **3** (01 2007). https://doi.org/10.1504/IJSPM.2007.015238
19. Singh, D., Padgham, L., Logan, B.: Integrating bdi agents with agent-based simulation platforms. Autonomous Agents and Multi-Agent Systems **30**(6), 1050âĂŞ1071 (Nov 2016). https://doi.org/10.1007/s10458-016-9332-x, https://doi.org/10.1007/s10458-016-9332-x
20. Uhrmacher, A.M., Weyns, D.: Multi-Agent Systems: Simulation and Applications. CRC Press, Inc., USA, 1st edn. (2009)
21. Uhrmacher, A.: Simulation for agent-oriented software engineering. In: Proc. of 1st Int'l. Conference on Grand Challenges for Modeling and Simulation, San Antonio, Texas, USA, 27-31 January. (2002)
22. Vincent, R., Horling, B., Lesser, V.: An agent infrastructure to build and evaluate multi-agent systems: The java agent framework and multi-agent system simulator. In: Wagner, T., Rana, O.F. (eds.) Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems. pp. 102–127. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
23. Zeigler, B.P., Kim, T.G., Praehofer, H.: Theory of Modeling and Simulation. Academic Press, Inc., USA, 2nd edn. (2000)

A Events and Activities

EvAgRCBegin / EvAgRCEnd	Reasoning cycle begin and end.
EvAgSenseBegin / EvAgSenseEnd	Sense stage begin and end.
EvAgDelBegin / EvAgDelEnd	Deliberate stage begin and end.
EvAgActBegin / EvAgActEnd	Act stage begin and end.
EvAgFetchPercept	Agent checking for percepts in the sense stage.
EvAgCheckMail	Agent checking for mails in the sense stage.
ActAgRC	reasoning cycle activity.
ActAgBetweenRC	activity between two subsequent reasoning cycles.
ActAgSense	Sense stage activity.
ActAgDel	Deliberate stage activity.
ActAgAct	Act stage activity.

Table 1: Agent EC events and activities concerning the reasoning cycle, and the sense stage in particular (excluding those involving agent communication and agent-environment interaction). The same names are used in the current JaCaMo-sim implementation (in particular, for each event a Java class with the same name is defined).

EvAgIntActMsgSend	A new send action has been performed (act stage).
EvCommSendMsgDispatch	The dispatch of new message has been requested to the Comm medium by a sender agent. This is the synchronous event (as defined in Section 3) created in the Communication EC corresponding to EvAgIntActMsgSend event occurring in the agent EC.
EvCommReceiveMsgDispatch	A message is ready to be delivered to some target agent. This event is caused by the EvCommSendMsgDispatch event.
EvAgNewMsgNotifier	A new message is asynchronously notified to an agent. This is the synchronous event on the agent side of the EvCommSendMsgDispatch event.
EvAgNewMsgReceived	The agent has received the message in the sense stage. This event is caused by the EvAgCheckMail event.
ActCommMsgDispatch	Activity devoted to deliver the message by the Communication medium.

Table 2: Events and activities involved in agent communication.

EvAgExtActRequest	An external action request, to execute an operation on some artifact.
EvWspActDispatch	A new request of action be dispatched by the workspace to the target artiact. This is the synchronous event created in the workspace EC corresponding to EvAgExtActRequest event occurring in the agent EC.
EvWspNewOpToExec	Event representing a new op execution to be served inside the workspace. This event is caused by the EvWspActDispatch event.
EvArtOpEnqueued	Event generated when the operation request has been enqueued in the artifact. This is the synchronous event on the artifact side of the EvWspNewOpToExec event on the workspace side.
EvArtOpExecBegin	Event representing the beginning of operation execution on the artifact side. This event is caused by the EvArtOpEnqueued event.
EvArtObsStateEvent	Event representing a change to the observable state of the artifact (including the generation of a signal).
EvWspObsStateDispatch	Event representing a obs state event to be dispatched from an artifact. This is the synchronous event on the workspace side of the EvArtObsStateEvent event on the artifact side.
EvArtOpExecEnd	Event representing the end of operation execution on the artifact side.
EvWspActionEventDispatch	Event representing an action event to be dispatched from an artifact, representing action/operation success or failure. This is the synchronous event on the workspace side of the EvArtOpExecEnd event on the artifact side.
EvWspPerceptDispatch	Event representing a new percept ready to be dispatched to an agent, either representing a new observable event or an action event. This event is caused either by the EvWspActionEventDispatch event or the EvWspObsStateDispatch event.
EvAgNewPerceptNotified	A new percept notified to the agent percept queue. This is the synchronous event of with EvWspPerceptDispatch on the workspace side.
EvAgBelUpdatedFromPercept	Agent updating a belief from a percept (sense stage). This event is caused by the EvAgNewPerceptNotified event (about a new observable state event).
EvAgExtActResult	Agent perceiving the success or failure of an action previously requested (sense stage). This event is caused by the EvAgNewPerceptNotified event (about a new action event).

Table 3: Events involved in agent-environment interaction.

ActWspOpDispatch	Activity on the workspace side to deliver a new operation to be executed on an artifact.
ActArtOpDispatch	Activity to deliver an enqueued operation to an artifact to be executed.
ActArtOpExec	Activity concerning the execution of an operation.
ActArtObsStateDispatch	Activity to dispatch of a new observable event to the worskpace.
ActArtActionEventDispatch	Activity to dispatch an operation/action result to the workspace.
ActWspActEventToPerceptDispatch	Activity on the workspace side to deliver an operation/action result as a percept to the agent that executed the action.
ActWspObsStateToPerceptDispatch	Activity on the workspace side to deliver an new observable event as a percept to an agent observing the artifact.

Table 4: Activities involved in agent-environment interaction.