Communication- and Computation- Bounded Agents in Multi-Agent Simulations

Michal Čáp, Jiří Vokřínek, and Antonín Komenda

Agent Technology Center
Gerstner Laboratory – Agent Technology Center
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
Technická 2, 16627 Praha 6, Czech Republic
{cap,vokrinek,komenda}@agents.felk.cvut.cz

Abstract. This paper proposes a mechanism for simulating limited communication bandwidth and processing power available to an agent in multi-agent simulations. Although there exist dedicated tools able to simulate computer networks, most multi-agent platforms lack support for this kind of resource allocation. We target such multi-agent platforms and offer an easy method to implement the missing functionality by the agent designer. The introduced method assigns two additional message buffers to each agent, which are used to (i) limit the number of messages an agent is able to send in one simulation round, and (ii) limit the number of messages an agent is able to process in one simulation round.

1 Introduction

Multi-agent simulation is a simulation of (physical) entities controlled by agent-based models.[1] Such a simulation can be used in two different context. Economist, biologists, and social scientists often use multi-agent simulation as a specific simulation model in which the designer explicitly defines the process by which agents (actors) in the simulation make their decisions. Complex behaviour of the simulated system emerges from the micro-behaviours of its individual agents. Many tools exist that facilitate development of such simulations, e.g. NetLogo [8] or MASON [2].

An alternative view sees a multi-agent simulation as an evaluation tool for agent based systems. Under this interpretation, the simulation is used to evaluate/predict the applicability of the given agent-based solution in the real world. An example of such a platform is AgentFly simulation [6], which simulates the behaviour of a number of aircraft in U.S. national airspace in order to evaluate the performance of agent-based deconfliction algorithms developed within

[1] In the following text, we will call an entity controlled by an agent-based model simply an agent.

AgentFly project [7]. The work presented in this paper is mainly relevant to this interpretation of the term.

The two following problems typically arise in multi-agent simulations. Firstly, the computational and communication resources available to each agent are distributed unequally over the multi-agent system. Often, agents residing on a fast computer can use more computational power than the entities residing on a slow computer. Secondly, the computational power and communication bandwidth an agent can use in a simulation do not reflect the capabilities of the hardware and the communication channel the agent would have used in the real world. There are domains in which such discrepancies between simulated environment and real environment seriously harm predicative value of a simulated deployment of the multi-agent system.

The ability to constrain the execution of individual agents is a fundamental requirement especially in distributed or asynchronous simulations, where each agent may run on a different computer. In such a situation, the simulation engine has to guarantee a fair distribution of processing power among the agents even if the agents run in non-deterministic, heterogeneous computational environments connected by dissimilar communication links.

1.1 Agents in Simulation

As mentioned earlier, this paper focuses on multi-agent simulations that are used to evaluate performance of agent-based systems. Therefore, we consider an agent as a system that interacts with a simulated environment. We assume that an agent is defined in terms of the agent's internal state and a sense-reason-act cycle that continuously updates the agent's internal state. The agent perceives the state of the environment using its sensors and it makes interventions in the environment using its effectors. The agent interacts with the other agents via the environment or using the agent communication channel provided by the agent platform. The described model is depicted in Figure 1. In the scope of this article we see agents as software entities developed on top of some (existing) agent platform.

2 Related Work

There is a number of platforms that facilitate the development of agent based systems. Agent development platforms mainly provide agent life-cycle management, message transport support, and debugging tools. Some of the platforms also provide support for distribution of the multi-agent system across different computers. An exhaustive list of agent development frameworks can be found in [10]. An example of a popular, Java based agent development framework of this kind is JADE [9]. To our knowledge, however, none of the existing agent frameworks provides the support for bounding the communication bandwidth and/or processing power available to an agent.

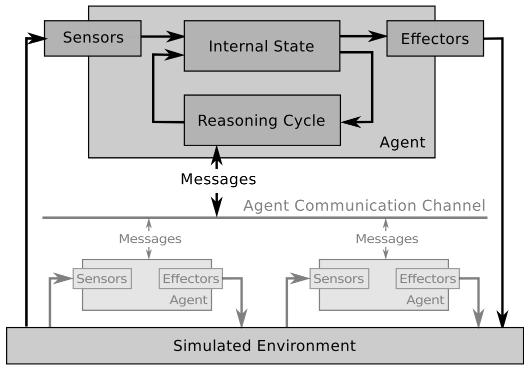

Fig. 1. Multi-agent System Architecture

Looking from another perspective, the research area of network simulators is a well established one. Several software tools have been developed (e.g. ns-3 [11] or CORE [12]) able to faithfully emulate the behaviour of a modelled network, including the parameters such as link throughput, latency, error rate etc. Although these simulators provide high fidelity simulation of network properties, they are by no means agent-oriented. An integration with one of the agent enabling platform seems to be a non trivial problem. However, there is a recently started project AHOY [13] that aims to employ network simulators of this kind in multi-agent simulations.

The mechanism we propose is a pragmatic one. It allows a programmer to choose an agent platform and extend it with a simple communication- and computation- bounding functionality with relatively little effort.

3 Message Buffer Based Approach

The proposed mechanism is based on the idea of limiting the number of messages an agent is able to receive and send in a given period of time. The method is implemented in the message transport layer of an agent platform and controls how the individual agents dispatch and process messages.

We assume the environment to be a round-based simulation advancing in constant, predefined time steps (simulation rounds). The starting time point of a simulation round is called *a round tick*. The main principles of our method are the following:

– **Bounding Communication Bandwidth**: The number of messages an agent is allowed to dispatch during one simulation round represents the bandwidth of its communication channel. By limiting this number we are able to control the maximum communication bandwidth each individual agent can use.
– **Bounding Processing Power**: First, we make two simplifying assumptions: (i) we assume a reactive agent that performs computation only as a

consequence of a received message, (ii) we assume that all types of messages trigger the same amount of computation within the agent. Consequently, the number of messages delivered to an agent in one round represents the amount of computation the agent performs during the round. By limiting the number of messages an agent receives, we can control the amount of processing power an agent consumes in one simulation round.

If we apply the two introduced principles, no agent can disturb the simulation by exploiting more processing power (e.g. because it is running on more powerful hardware than others) and overloading the other agents' message buffers with (e.g. request) messages.

The mean simulated communication bandwidth $\bar{r}$ available to an agent is directly proportional to the number of messages the agent can transfer in one simulation round:

$$\bar{r} = \frac{m_o \bar{s}_m}{t}, \tag{1}$$

where $\bar{s}_m$ is the mean message size in the system, and t is the duration of one simulation round (in simulated time), m_o is the number of messages the agent can send in one simulation round. Parameter m_o is configurable and can be used to set the communication bandwidth available to a particular agent. The processing power allocated to an agent can be derived in a similar way:

$$\bar{p} = \frac{m_i \bar{i}_m}{t}, \tag{2}$$

where $\bar{p}$ denotes the mean simulated computational power available to an agent, $\bar{i}_m$ is the mean number of instructions an agents needs to process one message, and t is the duration of one simulation round. The configurable parameter m_i represents the maximum number of messages an agent can process in one simulated round, which can be used to set the maximum processing power allocated to the agent.

The parameters m_i and m_o can be set independently for each agent to set its communication and processing bounds. The parameters are computed using the mean message size and the mean number of instructions to process a message. Clearly, the use of mean values might yield unintended behaviour in some situations (e.g. a trivial problem is assumed to consume the same amount of processing time as a complex problem). Nevertheless, in all our testing scenarios, the method provided us with satisfactory results.

3.1 Multi-Agent Simulation Architecture

A multi-agent simulation typically consists of two parts. The first part is an environment simulator that simulates the state of the world. The second part are the individual agents acting in the simulated world. We assume that the agents are developed on top of a distributed agent platform, which is extended with the synchronization mechanism explained in the following section and the buffered communication as presented in Section 3.3.

3.2 Synchronization Mechanism

Our communication and computation bounding method requires the simulation to be synchronized with the reasoning process of the individual agents. The following mechanism ensures that the simulation will not proceed to the next simulation round until all the agents have finished their message processing.

The mechanism is based on two signals that the agents use to control the counter of pending messages maintained by the simulation. Whenever an agent sends a message, it also sends an *increase counter signal (ICS).* When an agent processes a message, it sends a *decrease counter signal (DCS).* The simulation round cannot be finished (i.e. next round tick generated) if the number of DCS generated during the current round is lower then number of ICS. Such a situation occurs if (i) some agents still process messages or (ii) some messages are still being transferred.

Further, each round tick signal is followed by an ICS send by all agents. This implies that the round can not be finished before all agents process the signal and all the other messages exchanged during this round. To maintain causality and consistency of the synchronization mechanism, we need to ensure that (i) the order of the control signals sent by an agent is maintained, and (ii) an agent sends one DCS after it has processed all messages, i.e. all messages to other agents and the corresponding ICS were sent before it.

In Figure 2, we show an example run of a simulation consisting of four rounds. The round ticks are labelled in the *simulation time* (i.e. the time observed by the simulated entities). The *extra time* is used to virtually increase the communication bandwidth and/or processing power for the agents. The processing power is directly proportional to the amount of extra time, since as the extra time increases, the time an agent can use for processing increases as well. In other words, more physical time for processing means more computation power from the perspective of the agent, provided that the simulation time is stalled.

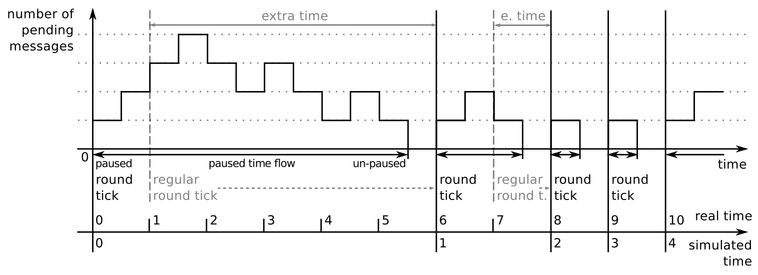

Fig. 2. Counting the Pending Messages

In the presented approach, the amount of extra time is controlled by the counter of pending messages c (see Figure 3). Each ICS sent by an agent increments the counter of pending messages $c_{k+1} = c_k + 1$ (in Figure 2 denoted by a rising edge) and analogically, each DCS decreases the counter $c_{k+1} = c_k - 1$ (in Figure 2 denoted by a falling edge). If $c > 0$, then there are unprocessed messages in the system. In such a situation, the simulation time is paused until $c = 0$. Only then the next round tick can be generated.

In its basic form, the counter of pending messages pauses the simulation until all pending messages are processed, which corresponds to a situation in which all agents have infinite communication and processing bounds. That is, the simulation will wait for a conversation of arbitrary length to finish, even for an infinite one. Since we want to simulate real machines and real communication channels, we want to be able to interrupt conversations after a predefined number of messages and finish the simulation round in finite real time.

This leads us to the concept of three message buffers which allow us to reliably interrupt a conversation between two agents at any point.

3.3 Buffered Communication

In the classical model, an agent uses one message queue (buffer), in which it stores all incoming messages. These messages are then one by one processed by the agent's deliberation cycle. Message sending is typically realised by remote addition of the message to the recipients message queue.

In three message buffers concept, an agent uses two additional queues for message processing. The first additional queue buffers the incoming messages, the second queue buffers the outgoing ones. In this scheme, a sent message may be buffered before it is delivered to the recipient's message queue. Similarly, a received message may be stalled in the agent's incoming message buffer before it gets processed by the agent[2]. The two additional message buffers let us safely interrupt any conversation between two agents.

Figure 3 shows the principle of three message buffers based communication. The message processing algorithm is the following:

- When an agent receives a round tick, it checks the outgoing message buffer and dispatches the messages (it sends one ICS for each message) until the number of sent messages during this round m'_o reaches m_o or until the buffer is empty. Then, the agent checks the incoming message buffer and processes the messages until the number of messages processed during this round m'_i reaches m_i. Finally, after this step is done, the agent sends DCS.
- When the agent receives a message in its main message queue[3], it performs the following. If the number of messages processed during this round m'_i is smaller than m_i, then it increases m'_i by one and processes the message as usual. Otherwise, the message is added to the incoming message queue. The agents sends DCS in both cases.

[2] The main message queue and the incoming message queue are kept separate, so that the concept can be implemented as an extension of existing multi-agent system or platform.

[3] Shown as the widest buffer connected to ACC in Figure 3.

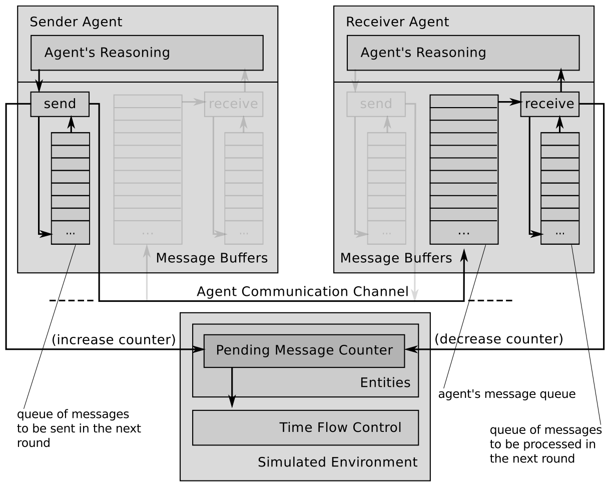

Fig. 3. Three Message Buffers Communication

– When the agents attempts to send a message, we do the following. If the number of messages sent during this round m'_o is smaller than m_o, the agent sends ICS, dispatches the message and increases m'_o by one. Otherwise, the message is buffered in the outgoing message queue.

We like to note that the presented mechanism can be implemented transparently in the message transport layer of most agent platforms. The code specifying the agents running on the platform may stay intact.

4 Illustration

In this section, we present an illustrative example that explains the proposed concept in an intuitive form. In our simple multi-agent system, there are only eight messages and two agents. The roles of the two agents are clearly distinguished. The sender agent sends messages, while the receiver agent receives them. The outgoing message limit is set to $m_o = 5$ messages. The incoming message limit is set to $m_i = 3$ messages.

The values of m_o and m_i were chosen based on the following motivation. Let's assume we want to simulate the communication throughput of human speech. The average sentence (message) is $\bar{s}_m = 90$ bytes long. The communication bandwidth of human speech is approximately $\bar{r} = 7.5$ bytes per second (which is about 12 words spoken in 10 seconds). Further, we want to perform the simulation with the granularity of 1 minute, therefore $t = 60$ seconds. Using Formula 3, we can compute the maximum number of outgoing messages in a simulation round:

$$m_o = \frac{\bar{r}t}{\bar{s}_m} = 5. \tag{3}$$

We can control the processing power in a similar manner. Let's assume that the sentences contain arithmetic exercises, where each of them contains on average $\bar{\bar{i}}_m = 2$ arithmetic operations (instructions). Each operation can be solved by a human in 10 seconds ($\bar{p} = 1/10$). Using Formula 4, we can compute the number of incoming messages:

$$m_i = \frac{\bar{p}t}{\bar{\bar{i}}_m} = 3. \tag{4}$$

The proposed scenario running on a single-core 2.0GHz computer (for simplification, let us assume one operation can be done in one tact) would be theoretically able to simulate $20 \cdot 10^9$ simulated seconds in one real second. The result is depicted in Figure 4 as a sequential diagram.

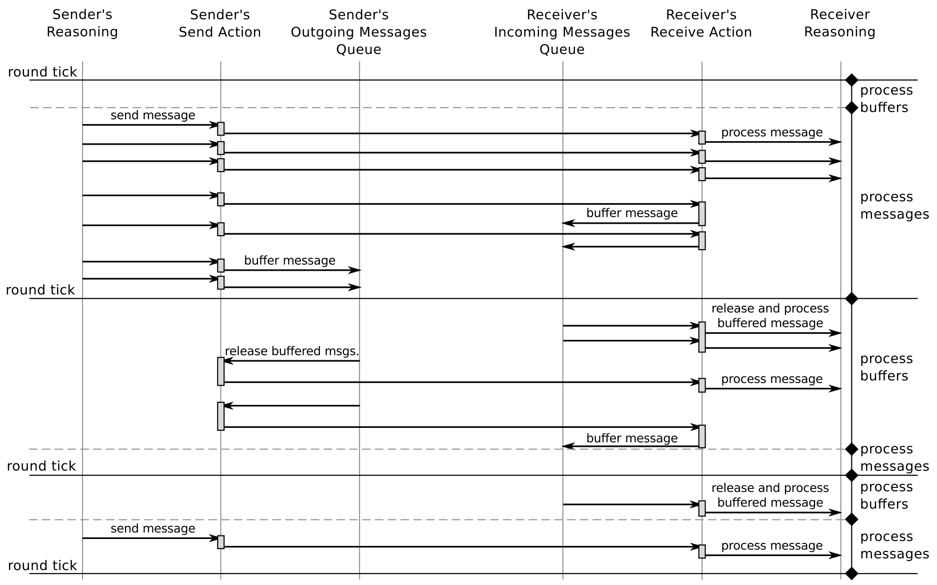

Fig. 4. A sequence diagram showing communication based on three message buffers for one sender and one receiver, where $m_o = 5$ and $m_i = 3$

The experiment has illustrated a typical message passing process in a system that employs the communication model based on three message buffers and the synchronization model based pending message counter.

5 Evaluation

The presented mechanism was applied in i-Globe project[14]. In this project, we explored the methods of mixed-initiative planning and decision making [5]. We created a multi-agent simulation of geographically distributed mobile units operating on an hostile island. The agents (ground units) were programmed to fulfil strategic goals that were generated for them by mid and long-term planners. The agents used their tactical planners to generate their tactical plans. The execution of each agent's individual plan was simulated in a simulated environment.

A typical communication message between two agents contained one task request ($\bar{i}_m = 1$). The computation time of a planner used in the scenario was 78ms ($\bar{p} = 1/0.078 \doteq 12.8$). The duration of a simulation round is 100ms, which corresponded to $t = 10$s simulated time. Using Formula 5, we computed the value of m_i:

$$m_i = \frac{\bar{p}t}{\bar{i}_m} = \frac{12.8 \cdot 10}{1} = 128. \tag{5}$$

In this scenario, an agent's communication bandwidth was not limited ($m_o = \infty$). Instead, we wanted to maintain a fair distribution of processing power. The adapted system succeeded to eliminate the manifestations of unfair processing power distribution (e.g. some of the agents being overloaded by task request from the faster processing agents).

Further, we tested the mechanism in a MANET (mobile ad-hoc network) scenario. The scenario involved a number of mobile routing units that had to establish and maintain network links between them. A simulation round was 100ms long and the communication bandwidth has been set to 54 Mbps. The mean size of one message was experimentally measured to be 200 bytes. Using Formula 6, we computed the value of m_o to be 3375.

$$m_o = \frac{\bar{r}t}{\bar{s}_m} = \frac{6.75 \cdot 10^6 \cdot 0.1}{200} = 3375. \tag{6}$$

We have successfully used the framework to limit the bandwidth available to each agent.

6 Conclusion

We have designed a mechanism that allows for implementation of agents that are communication and computation bounded. Using this mechanism, a multi-agent simulation can be distributed over a number of different computers, without worrying about some of the agents having communication or processing advantage over the others. Moreover, the mechanism provides robust round tick synchronization for distributed asynchronous simulations.

We designed the mechanism in such a way that it can be implemented on top of most existing agent platforms. The agents themselves may stay intact.

Since the values of m_i and m_o are set in the design time and based on the average message size and the average amount of processing a received message triggers, potential future work would be to generalize the approach in the direction of run-time analysis of the messages being sent. Then, we would be limiting the maximum number of outgoing messages based on the sum of their actual lengths instead of relying on their mean length. Similarly, our current approach to computational power bounding is based on a strong assumption that the amount of computation an agent performs is proportional to the number of messages it receives. To relax this assumption, we envision a generalised mechanism that would count the number of instructions the agent performs during one simulation round. As soon as the count exceeds a predefined threshold, the simulation engine will stop the agent's execution thread.

Acknowledgements The presented work was supported by the U.S. Army Communications-Electronics Research, Development and Engineering Center grant no. W911NF-08-1-0521, Czech Republic Ministry of Education, Youth and Sports, grant no. MSM6840770038, the Grant Agency of the Czech Technical University in Prague, grant no. SGS10/189/OHK3/2T/13 and the research programme no. 1M0567 funded by the Ministry of Education of the Czech Republic.

References

1. Wang, F., Turner, S.J., Wang, L.: Agent Communication in Distributed Simulations. In: Multi-Agent and Multi-Agent-Based Simulation, Joint Workshop MABS 2004, New York, NY, USA, July 19, 2004.
2. Luke, S., Cioffi-Revilla, C., Panait, L, Sullivan, K.: MASON: A New Multi-Agent Simulation Toolkit. In: Proceedings of the 2004 Swarmfest Workshop, 2004.
3. Gulyas, L., Bartha, S.: FABLES: A Functional Agent-Based Language for Simulations. In: Proceedings of The Agent 2005 Conference on: Generative Social Processes, Models, and Mechanisms. Chicago, IL: Argonne National Lab., 2005.
4. Sislak, D., Rehak, M., Pechoucek, M.: A-globe: Multi-Agent Platform with Advanced Simulation and Visualization Support. In: Web Intelligence. IEEE Computer Society, 2005.
5. Komenda, A., Pěchouček, M., Bíba, J., Vokřínek, J.: Planning and Re-planning in Multi-actors Scenarios by means of Social Commitments. In: Proceedings of Workshop on Agent Based Computing V, 2008.
6. Šišlák, D., Volf, P., Jakob M., Pěchouček, M.: Distributed Platform for Large-Scale Agent-Based Simulations. In: Agents for Games and Simulations, LNAI 5920, 2010
7. Šišlák, D., Volf, P., Pěchouček, M.: Agent-Based Cooperative Decentralized Airplane Collision Avoidance. IEEE Transactions on Intelligent Transportation Systems, 2011
8. Wilensky, U., Rand, W.: An introduction to agent-based modeling: Modeling natural, social and engineered complex systems with NetLogo. Cambridge, MA: MIT Press.
9. Bellifemine, F., Bergenti, F., Caire, G., Poggi, A.: JADE - a Java agent development framework. In: Multi-Agent Programming: Languages, Platforms and Applications. Kluwer, 2005.
10. AgentLink - Agent Software, http://eprints.agentlink.org/view/type/software.html
11. ns-3 Network Simulator, http://www.nsnam.org/
12. Ahrenholz, J., Danilov, C., Henderson, T.R., Kim, J.H.: CORE: A real-time network emulator. In: Military Communications Conference, 2008. IEEE 2008
13. Ahoy Project, http://ahoy.googlecode.com
14. Komenda, A., Vokrinek, J., Pechoucek M., Wickler G., Dalton, J., Tate, A.: I-Globe: Distributed Planning and Coordination of Mixed-initiative Activities. In: Proceedings of Knowledge Systems for Coalition Operations (KSCO 2009).