Rafael H. Bordini, Louise A. Dennis and Yves Lespérance (eds)
Pre-proceedings of the 7th International Workshop on Engineering Multi-Agent Systems (EMAS 2019)
13th-14th May 2019, Montreal, Canada
cgi.csc.liv.ac.uk/~lad/emas2019/accepted/EMAS2019_paper_22.pdf

An Architecture for Integrating BDI Agents with a Simulation Environment

Alan Davoust[2,1], Patrick Gavigan[1], Cristina Ruiz-Martin[1], Guillermo Trabes[1,3], Babak Esfandiari[1], Gabriel Wainer[1], and Jeremy James[4]

[1] Carleton University, Ottawa, Canada
patrickgavigan,cristinaruizmartin,guillermotrabes,babak,gwainer@sce.carleton.ca
[2] Université du Québec en Outaouais, Gatineau, Canada
alan.davoust@uqo.ca
[3] Universidad Nacional de San Luis, Argentina
[4] Cohort Systems
jjames@cohortsys.com

Abstract. We present Simulated Autonomous Vehicle Infrastructure (SAVI), an open source architecture for integrating Belief-Desire-Intention (BDI) agents with a simulation platform. This approach decouples the development of complex multi-agent behaviours from the development of simulated environments to test them in.
We identify and address the *impedance mismatch* between modelling and simulation and BDI systems. Our approach avoids linking the environment's simulation time step to the agents' reasoning cycles: if the agents' reasoning is slow, perhaps due to expensive computations, the simulation will continue unaffected. Conversely, SAVI also prevents the reasoning from running faster than the simulation.
Both of these situations should be impossible for simulated environments that are meant to approximate the dynamic and continuous time nature of the real world. This is accomplished by running the simulation cycles and the agent reasoning cycles each in their own threads of execution, and managing a single point of contact between these threads. Finally, we illustrate the use of our architecture with a case study involving the simulation of Unmanned Aerial Vehicles (UAVs) following birds.

Keywords: Belief-Desire-Intention (BDI) · Modeling and Simulation · Architecture · Jason · AgentSpeak Language (ASL)

1 Introduction

Multi-agent systems are often designed to be embedded in highly dynamic environments. In these environments, the wide range of possible input signals may produce complex group-level behaviours which are difficult to accurately predict or to produce by design. During the development process, the behaviour of the agents must therefore be thoroughly tested in a controlled yet realistic environment before the system can be deployed. In this research, we are concerned with the development of agents using the Belief-Desire-Intention (BDI)

paradigm [24], and of an appropriate simulated environment to test the agent system. The main challenge in this task is the lack of frameworks to appropriately handle both the development of complex cognitive agents and of a realistic simulated environment[27, 1].

Existing BDI frameworks, such as Jason [14, 10] and lightJason [4, 18], include simple environments that can be reused and extended, but these environments lack the sophistication and graphical capabilities of proper simulation platforms. Conversely, the field of Modelling and Simulation provides a set methodologies with their own simulation tools (e.g. the Discrete Event System Specification (DEVS) formalism [35] with simulators including CD++ [33] and PyDEVS [28], Agent Based Modelling (ABM) with tools including Repast[22] or NetLogo [34]). It also provides domain specific simulation platforms for communication networks (e.g. OMNET++ [32]), traffic simulation (e.g. MITSIMLab [7], Microscopic Traffic Simulator [17]), and other domains. However, these are poorly suited for modelling complex cognitive processes [1]; in particular, they do not provide any support for techniques such as BDI.

As a result, the main approaches to integrating these two pieces involve either writing custom simulation code in a BDI framework, or custom BDI support in a simulation platform, or finally integrating two separate, mature frameworks from the two areas, with a considerable *impedance mismatch* problem [27]. By this term we refer to the conceptual and technical issues faced when integrating components defined using different methodologies, formalisms or tools.

Our work follows the third approach, and aims to integrate BDI agents with a simulated environment. Our main contribution is Simulated Autonomous Vehicle Infrastructure (SAVI), an architecture that seamlessly connects the Jason BDI framework [14, 10] with a simulation environment developed using Processing [13], addressing several key elements of the impedance mismatch problem.

In particular, our architecture decouples the agents from the simulation environment, making it easy to develop them independently, and allowing them to run as separate processes interacting in an asynchronous manner. This avoids linking the environment's simulation advances to the agent's responses: if the agent is for some reason slow (e.g. due to expensive computation), the simulation will continue unaffected, making the transition to a natural environment more realistic.

The rest of the paper is organized as follows: in section 2 we provide a summary of background followed by related work in section 3. This is followed by a definition of the proposed SAVI architecture is in section 4. In section 5 we describe a case study applying our architecture. Finally, we conclude in section 6 followed by a brief section on future work.

2 Background

In this section we present the BDI paradigm for developing multi-agent systems and discuss different approaches to develop a BDI agent system in a simulated environment.

2.1 Belief-Desire-Intention Architecture

The BDI architecture was introduced by Bratman and others in the 1980s[11] as a way to develop complex intelligent and autonomous agents. In this method, agents have a set of beliefs, stored in a *belief base* about their state as well as the state of their environment. They can perceive their environment to update these beliefs. These agents also have goals, or *desires*, that they need to achieve. The agents also have a set of plans that they can execute, stored in a *plan base*. The plans can involve updates to the belief base, or actions that the agent can apply to the environment. When an agent reasons about its *beliefs* and desires and selects an appropriate plan to execute, this plan becomes an *intention*. As the agent executes these plans, the agent can drop intentions based on changes in their beliefs if they are no longer achievable due to some change in the environment. The execution cycle for a BDI agent, from perception to action, is called the *reasoning cycle*.

BDI architectures have become especially relevant with the development of autonomous vehicles, in particular, self-driving cars (see [25] for example). However, as testing the vehicles' decision-making in a real-life setting is challenging, it is crucial that they can be developed in a realistic simulated environment.

2.2 Simulation Requirements for Multi-Agent Systems

A multi-agent system is situated in an environment (real or simulated) which the agents can perceive through different types of sensors (e.g. a camera or lidar sensor), and modify through actuators (e.g. moving to a new location, picking up an object).

An important property of the environment is whether it is *static* or *dynamic* [26]: a static environment contains only static objects that remain fixed and unchanging (the agents change only their internal state); a simple dynamic environment changes over time, but only due to the agents' actions; and finally a complex dynamic environment changes over time, due to the agents' actions but also due to other external factors, including natural phenomena, and agents outside of the considered agent system (e.g. humans).

The actions performed by the agents and the changes to the environment, either in response to the agents' actions or due to external factors, update the state of the system over time. There are several ways to model time in a simulation. One approach, Discrete-Event Modelling [5], updates the model's state variables every time an event occurs, and allows the model (or each sub-model of a composite model) to schedule its next state change, at any time. This allows arbitrarily fine-grained precision along the time axis. In an alternative approach, Discrete Time Modelling, the state of the simulation is updated at discrete points in time. The difference between a point in time and the next one is called the *time step*. This approach is well suited for applications where the system state changes very quickly or many events happen in a short period of time.

3 Related Work: Simulated Environments for BDI systems

There are three main approaches to simulate the environment of a BDI agent system [27]: the simulation can be built into the BDI engine, or else an existing simulation platform can be extended to support BDI, or finally a simulation platform can be connected to a BDI engine.

3.1 Simulation within MAS development platforms

Several agent development platforms have basic built-in simulation capabilities; this includes BDI platforms such as Jason.

In [9], the authors provide their own custom simulation environment for BDI agents. Another such custom simulation is [31], which includes *heavyweight agents* (i.e. agents with very advanced reasoning) combined with *lightweight agents* (i.e. agents that only reacts to their environment). However, if we compare these custom environments to other simulator platforms such as Repast, their features and capabilities are very limited. Typically, they are meant to support simple dynamic environments (which only change according to the agents' actions), and do not explicitly model time.

3.2 Modelling cognitive processes in simulation platforms

The second approach is to model the cognitive capabilities of agents (BDI, in our case) on standard simulation platforms.

Established ABM modeling tools (such as Netlogo or Repast) are not meant to directly model complex agent behaviours or realistic physical systems: their strength is rather in modelling the behaviour of complex systems as the emergent result of very simple interacting agent models. Similarly, general-purpose simulation systems (e.g. Mason [19]) and formalisms (e.g. DEVS) do not have built-in capabilities to model cognitive processes (such as the basic machinery of the BDI paradigm). However, there have been several attempts to build models of cognitive processes using modelling and simulation formalisms, in particular the DEVS formalism.

Several projects have implemented BDI reasoning with the DEVS formalism [30, 29, 1, 36].

JAMES [30, 29] is a Java Based agent modeling environment for simulation, to be used as test beds for multi-agent systems. It allows the execution of agents in distributed environments. In JAMES, an agent is represented as a DEVS atomic model, where its autonomous behaviour is represented by the internal function and the perceptions are represented through the external function. The actions of the agent in the environment are represented as the output function. In JAMES, the BDI architecture is incorporated in the internal state of the atomic model. Because in ABM, new agent are usually created and destroyed during the simulation, the authors also introduced Dynamic DEVS (DynDEVS).

The proposals of [1] and Zhang et al. [36] are very similar to the JAMES approach. The former use the classic dynamic DEVS (DS-DEVS) introduced by Barros [6] on a platform targeted at the simulation of agriculture called RECORD[8], whereas the latter implement implement PRS[15], a BDI-based reasoning architecture, on the D-SOL simulation platform[16]. Another model of cognitive processes, ACT-R, has also been modelled with DEVS[20].

DIVAs [3] is a simulation platform for dynamic and open environments that includes some machinery for agents' cognitive processes, including base classes to implement agent knowledge, tasks and plans. However, it is unclear whether this system uses an established simulation formalism or an established cognitive reasoning model.

3.3 Connecting simulation platforms and cognitive reasoning engines

A third approach, the one presented in this paper, aims to couple a mature platform for developing cognitive agents with an existing simulation platform. This can provide an improved modeling capability for simulations that involve complex agent behaviors. The main existing work in this direction [23, 27] is an integration of the commercial JACK platform [2] (for BDI agents) with the Repast agent based simulation software [21]. This is then generalized to an architecture that can accommodate wider range of ABM platforms and a wider range of platforms for modelling cognitive agents. The main weakness we see in this architecture is that it involves synchronizing the discrete-time simulation steps with the reasoning cycles of the cognitive agents. This implies that increasing the simulation granularity will also increase the relative speed of the agent's reasoning, since agents will be able to reason and act at much shorter time intervals of the simulation. In contrast, our approach allows the reasoning cycles to be decoupled from the simulation clock, and potentially get left behind by fast processes happening in the simulation.

Our approach is similar, although we have chosen Processing [13] as a simulation environment rather than an ABM tool. In our view, ABM modelling tools (such as Netlogo or Repast) are poorly suited to model complex dynamic environments. The agent-based modelling approach tends to model the entire system of interest (including physical systems deprived of any agency) as a system of (numerous) interacting agents. When the environment includes a small number of complex cognitive agents and a small number of (potentially complex) physical systems, other modelling methodologies appear more appropriate. ABM platforms also do not typically support discrete-event simulation.

Our choice of Processing is motivated by its powerful built-in visualization capabilities, and the option of using discrete-event simulation (although at this point our simulations are all discrete-time). A powerful graphical interface is useful for the demonstration of real scenarios to a non-technical audience. This is specially important for our use case, a military application where we need to test the resilience of the BDI agents. We note that we are also investigating

the applicability of this architecture to other simulation platforms, including Mason [19].

4 SAVI Architecture

This section details the proposed SAVI architecture for integrating BDI agents with a simulated environment. Specifically, we will focus on solving the problems resulting from the impedance mismatch between BDI and simulation systems. First, we introduce our framework setup, and briefly discuss the impedance mismatch problems. Then we introduce the open source SAVI architecture [12] and explain how these problems are addressed.

4.1 Setup

Our overarching problem is to connect an agent system built with a BDI framework to a model of the environment, designed using a framework appropriate for simulation. In our case, the BDI framework is Jason [10, 14], and our simulation runs in Processing [13]. Both are Java applications, which makes the integration manageable through direct method invocations, but the same approach would be feasible with any frameworks that expose the appropriate information via an external Application Programming Interface (API).

Our assumption is that the BDI agents are simply the reasoning engine (the *brain*) for agents with a physical presence in the simulated environment (e.g., drones or unmanned vehicles). In our case (see case study in section 5), these agent models are drones.

In order to connect the two "worlds" (i.e. simulation and BDI agents), the agent brains must receive perceptions of the world from the simulated agents, and send actions for the agent models to execute in the simulated world. Eventually, these agent brains will be connected to physical agents transmitting the same information as their simulated counterparts, and the goal is for the simulated behaviour to carry over into the real world.

4.2 Decoupling simulation and reasoning

In this context, one approach (adopted for example by Singh et al. [27]) is to use the discrete-time simulation process as a driver for the agents' reasoning: at each time-tick, update each simulated model, and invoke one reasoning cycle from the agent brains. This has the advantage of simplifying the integration of the two platforms, but it arguably comes at a significant cost in terms of a realistic simulation. In particular, it implies that changing the simulation time step (simply to change the granularity of the simulation), would directly affect the agents' *reasoning clock*: the agent reasoning will not be simulated more or less precisely, it will instead directly increase or decrease the agent's relative computational power, by allowing unbounded time for each decision. Pushed to the extreme, we might imagine for example an agent getting *lost in thought*

while computing intractable plans, and the world would then *wait* for the agent. Of course, this cannot happen in a *real world* environment: if an agent is lost in thought, the environment will continue to update while the agent performs its reasoning.

Therefore, our approach allows the agents' brains to run as their own processes (threads, more specifically), while the simulation will update on its own schedule. The two sides must now interact asynchronously, which brings several challenges (generically described above as the impedance mismatch between the two frameworks).

For one thing, actions may be initiated by the agent asynchronously, whereas the simulation system constrains changes to happen at fixed time steps. This is connected to a thread-safety issue, if both an agent process and the simulation process attempt to concurrently modify the environment.

More importantly, there is now a delicate balance to maintain between the simulation speed and the agents' reasoning speed. On one hand, if the agent reasons too fast, then it might repeatedly perceive an outdated state of the world and misinterpret the consequences of its latest action, which the simulation engine has not yet computed. The problem here is that this would not happen in the real world: there cannot be any delay between an action being initiated by the an agent in the real world, and this action initiating its effect on the environment. On the other hand, if the agent is slow and the perceptions from the environment come as messages, the agents might accrue a backlog of these messages, and again be attempting to act on an outdated perception of reality. In this case an agent being too slow to keep up with its environment is perfectly possible. However, the environment would not be sending overwhelming numbers of updates in the form of messages[5].

4.3 The SAVI Architecture

In order to address these challenges, we designed the SAVI architecture shown in figure 1.

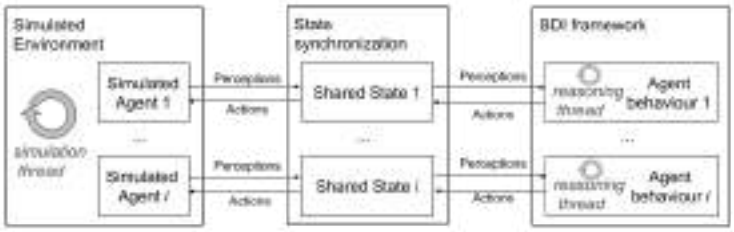

Fig. 1. Simulation and agent behaviour architecture.

[5] Of course, the perception infrastructure may do so, and again using the present architecture to implement that interface could solve the problem.

This architecture uses three main modules for implementing the interface between the BDI agents and the simulation infrastructure. These include the *Simulated Environment* module, the *State Synchronization* module, and the *Agent Behaviour* module. To ensure that the simulation time step is independent of the agent reasoning cycle rate of each of the agents, the simulation engine and each individual agent's behavioural model run in separate threads of execution.

The Simulated Environment module is responsible for providing the simulated environment as well as a simulated model of the agents' physical presence in that environment. This includes all movements and interactions of the agents. In our case, this module also provides a visualization of the environment for monitoring the simulation.

Individual agents perceive this simulated environment as well as their own properties, and perform actions. These interactions are mediated by the State Synchronization module. This module is responsible for ensuring mutual exclusion of the different execution processes over perception, messages, and actions being passed between the environment and agent objects in the Agent Behaviour module. This mutual exclusion is managed by ensuring that the variables representing the agent's perception are always calculated and set by the simulation side and only read by the behavioural models. Mutual exclusion of data between the simulation and agent threads is ensured using thread safe variables, and happens separately for each agent, meaning that there is no centralized bottleneck.

The agent behaviour module provides the implementation of the BDI based behaviour model. It receives environmental perceptions and messages from other agents via the State Synchronization module and responds by sending actions and messages back. These responses are determined using the BDI reasoning cycle, which runs as a separate thread of execution for each agent. This enables the updates to the environment to be decoupled from the execution time of the individual reasoning cycles of each agent. In addition, since the perceptions are represented as state variables to be read, as opposed to messages, there is never a backlog of perceptions waiting for the agent, even if the reasoning is slow.

Finally, in the case where the agent reasons faster than the simulation can update the environment variables, we ensure that the agent waits for new perceptions by implementing a *producer-consumer* pattern: if the simulation clock has not advanced since the previous reasoning cycle, the agent waits. For this purpose, the simulation engine timestamps every update of the state variables.

The effects of this speed coordination are illustrated by measurements of the simulation update speed and the reasoning speed, discussed in section 5.4.

5 Case Study

In this section, we describe our implementation of the SAVI architecture. We demonstrate the separation of the simulation from the implementation of the agent behaviours in BDI. We also show that we have overcome the impedance mismatch between these two techniques. Our case study scenario involves an airport safety patrol made up of Unmanned Aerial Vehicles (UAVs) chasing

migratory birds away from the airport. The implementation of SAVI, including this case study, are available as an open-source project [12].

5.1 Scenario

In this scenario, UAVs are controlled by BDI agents in order to chase migratory birds away from the airport property. The simulation environment represents the Ottawa airport area, the UAVs, and the different threats (migratory birds) that can appear in that area. Because the objective of the case study is to show our simulation architecture, and not necessarily to demonstrate the performance of complex behaviours, we use a simplified version of the problem where the UAVs' mission is simply to follow the different threats near the airport.

Each UAV perceives the environment through four sensors:

1. A Global Positioning System (GPS) receiver that provides the position of the UAV,
2. A velocity sensor that indicates the UAV's speed and direction of travel,
3. A camera that can see nearby threats and other UAVs up to a maximum range,
4. A clock.

Each UAV also has a set of simple actions related to moving in the environment. These include:

1. Turning to the left,
2. Turning to the right,
3. Activating a thruster to move forward,
4. Deactivating a thruster to stop moving.

The behavior of the UAVs in this simulation is defined in AgentSpeak Language (ASL) as follows:

- When the UAV does not perceive any threats, the UAV stops and keep turning until a threat is perceived.
- When the UAV perceives threats, it turns to face the nearest one and then follows it.

5.2 Implementation

The simulation is built using Processing [13], which handles the set-up and discrete-time simulation as well as visualisation. The agent behaviour to be deployed in the UAVs is defined using the BDI paradigm, with behaviours written in the ASL and interpreted using Jason [10, 14]. These two components are integrated as described above, using our SAVI architecture. The threat behaviour is directly implemented in Java: each threat sets a random destination and then travels in a straight line from its actual position. Once they arrive, they choose a new random destination.

5.3 Testing

Our case study consists of two scenarios, which can be easily set up in the simulation environment's configuration file. In the first scenario, shown in figure 2, we simulated the Ottawa airport area with 10 bird threats. The area is patrolled by three UAVs which have a limited camera perception range, as shown in the figure by a semicircle. Objects that are visible to a UAV are shown with circles around them. In the figure we can see that all the UAVs have a threat within their camera range; however, there is a large area that is not observed by the UAVs.

Fig. 2. Scenario 1: Test bed with 3 UAVs with short perception distance and 10 threats.

In the second scenario, shown in figure 3, we simulated the Ottawa airport area with 15 threats. The area was patrolled by 10 UAVs with longer-range cameras, also represented by a semicircle in the figure. In the figure we can see that all the threats are perceived at least by one UAV. Likewise, all the UAVs are perceived by at least another UAV, however some areas are not covered.

5.4 Results

The key objectives of the SAVI architecture were to connect BDI agents to a simulation platform and resolve the challenges of impedance mismatch associated with this task.

Fig. 3. Scenario 2: Test bed with 10 UAVs with large perception distance and 15 threats.

Our simulations were successfully run at several different frame rates, and the agents were able to carry out their task, largely unaffected by these variations. This demonstrates the suitability of our SAVI architecture to integrate a BDI framework with a simulation platform.

As noted earlier, we wanted to ensure that the agents' reasoning cycle and the simulation cycles were decoupled, and that we could manage their relative speeds. In effect, we needed to be certain that the simulation environment does not wait for the agent reasoning cycle to complete prior to computing the next simulation step. Furthermore, it was also of concern to ensure that the agent cannot reason faster than the simulation rate.

In order to demonstrate the effects of our safeguards on the speed coordination, we ran the test scenarios discussed in this case study under various frame rates and measured the effective simulation and reasoning cycle periods. The results for this test are shown in figure 4. The frame rate used as a reference along the X axis is the *requested* simulation speed, which may not be achievable in a computationally intensive simulation. We therefore plot the effective simulation speed (measured by the effective period between two frames) and the effective duration of the agents' reasoning cycles.

The plot shows that the reasoning cycle and the simulation time step follow an identical time step for lower frame rates, below approximately 65 frames per second. At these slower simulation speeds, the agent must synchronize its speed to only reason on up-to-date environment perceptions. The decreasing simulation

time step also shows that the simulator is able to achieve the requested frame rate.

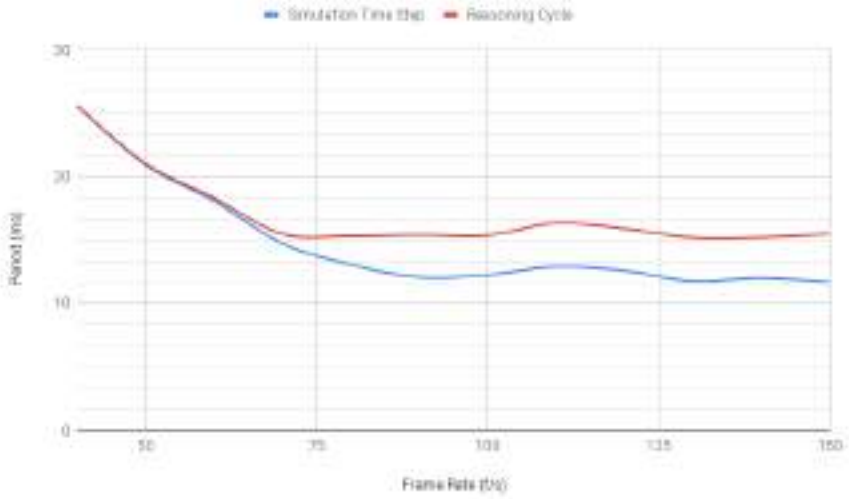

Fig. 4. Difference in simulation time step and reasoning cycle periods at different frame rates.

As the simulation frame rate increases, the simulation time step and the reasoning cycle period decrease but begin to diverge, and then they approximately stabilize at the highest speed that the simulator is able to achieve for this scenario. At a frame rate of approximately 75 frames per second we can clearly see the reasoning cycle period lag behind the simulation time step. This means that the agent is reasoning more slowly than the simulation updates proceed. The simulated environment is not delayed by a slow agent's reasoning cycle, and the agent does not develop a backlog of messages (which would affect the agents' performance and possibly the reasoning speed).

6 Conclusion

We have presented the SAVI architecture to integrate multi-agent systems developed using the BDI paradigm with a simulation platform. Our architecture decouples the execution of a time stepped simulation from the agent's reasoning processes, allowing them to run as separate processes interacting in an asynchronous manner. This contributes to a more realistic simulation by allowing the simulation of environment to proceed regardless of the agents' decision-making speed. However, we are nonetheless able to prevent the reasoning cycle from

executing faster than the simulation rate. This should not be possible in an environment that is meant to approximate dynamic environments in continuous time. These benefits of our architecture should make the transition to a natural environment more realistic. In addition, the decoupling of the two component frameworks makes it easy to develop them independently, and has allowed our team to successfully separate these two unrelated concerns during the development process. We have made our reference implementation available to the community as an open-source project[12].

Future Work

As the SAVI project is under active development, there are several developments planned as ongoing and future work. These include the connection of a human interface for providing some human command and control for the agent activities as well as a means for the agents to inform users of the state of the environment and as their state. We also want to expand SAVI so that it can run simulations with reasoners working at reasoning rates tied to the expected performance of real embedded reasoning systems. This could also mean connecting SAVI to agents that are loaded on real world hardware, where SAVI would stand in for a real world environment so that real world hardware can be tested in a controlled setting. Related to the goal of connecting to real world hardware, the SAVI project currently does not use any analogue sensing of the environment. Development is required in order to support the use agents with simulated analogue sensors which are connected to agents using BDI for higher level reasoning as part of a broader agent architecture.

References

1. Akplogan, M., Quesnel, G., Garcia, F., Joannon, A., Martin-Clouaire, R.: Towards a deliberative agent system based on DEVS formalism for application in agriculture. In: Proceedings of the 2010 Summer Computer Simulation Conference. pp. 250–257. SCSC '10, Society for Computer Simulation International, San Diego, CA, USA (2010), http://dl.acm.org/citation.cfm?id=1999416.1999447
2. AOSGroup: Jack. http://www.aosgrp.com/products/jack/, accessed: 2019-02-04
3. Araujo, F., Valente, J., Al-Zinati, M., Kuiper, D., Zalila-Wenkstern, R.: Divas 4.0: A framework for the development of situated multi-agent based simulation systems. In: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems. pp. 1351–1352. International Foundation for Autonomous Agents and Multiagent Systems (2013)
4. Aschermann, M., Kraus, P., Müller, J.P.: Lightjason: A bdi framework inspired by jason. In: Multi-Agent Systems and Agreement Technologies: 14th Europ. Conf., EUMAS 2016, and 4rd Int. Conf., AT 2016, Valencia, Spain, 2016. pp. 58–66. Springer International Publishing (2016)
5. Banks, J., Carson, J., Nelson, B., Nicol, D.: Discrete-Event System Simulation. Prentice Hall, 5 edn. (2010)

6. Barros, F.J.: Dynamic structure discrete event system specification: A new formalism for dynamic structure modeling and simulation. In: Proceedings of the 27th Conference on Winter Simulation. pp. 781–785. WSC '95, IEEE Computer Society, Washington, DC, USA (1995). https://doi.org/10.1145/224401.224731, http://dx.doi.org/10.1145/224401.224731
7. Ben-Akiva, M., Koutsopoulos, H., Toledo, T., Yang, Q., Choudhury, C., Antoniou, C., Balakrishna, R.: Traffic simulation with mitsimlab. In: Barceló, J. (ed.) Fundamentals of Traffic Simulation, pp. 233–268. Springer New York (2010). https://doi.org/10.1007/978-1-4419-6142-6_6, https://doi.org/10.1007/978-1-4419-6142-6_6
8. Bergez, J.E., Chabrier, P., Garcia, F., Gary, C., Makowski, D., Quesnel, G., Ramat, E., Raynal, H., Rousse, N., Wallach, D.: RECORD: a new software platform to model and simulate cropping systems. Farming System Design, Monterey, CA (2009)
9. Bordini, R.H., Hübner, J.F.: BDI agent programming in agentspeak using jason. In: Toni, F., Torroni, P. (eds.) Computational Logic in Multi-Agent Systems. pp. 143–164. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
10. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology). John Wiley & Sons, Inc., USA (2007)
11. Bratman, M.: Intention, plans, and practical reason, vol. 10. Harvard University Press (1987)
12. Davoust, A., Gavigan, P., Trabes, C.R.M.A.G., Esfandiari, B., Wainer, G., James, J.: Simulated autonomous vehicle infrastructure. https://github.com/NMAI-lab/SAVI, accessed: 2019-02-19
13. Fry, B., Reas, C.: Processing. https://processing.org/, accessed: 2019-02-16
14. Hübner, J.F., Bordini, R.H.: Jason: a java-based interpreter for an extended version of agentspeak. http://jason.sourceforge.net, accessed: 2019-02-16
15. Ingrand, F.F., Georgeff, M.P., Rao, A.S.: An architecture for real-time reasoning and system control. IEEE expert **7**(6), 34–44 (1992)
16. Jacobs, P.H., Lang, N.A., Verbraeck, A.: D-sol; a distributed java based discrete event simulation architecture. In: Proceedings of the Winter Simulation Conference. vol. 1, pp. 793–800. IEEE (2002)
17. Jaworski, P., Edwards, T., Burnham, K.J., Haas, O.C.L.: Microscopic traffic simulation tool for intelligent transportation systems. 2012 15th International IEEE Conference on Intelligent Transportation Systems pp. 552–557 (2012)
18. Kuper, C., Jetbrains, Müller, J.P., Spitzer, M., Tatasadi, E.: Lightjason. https://lightjason.org/, accessed: 2019-03-15
19. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: A multiagent simulation environment. Simulation **81**(7), 517–527 (2005)
20. Mittal, Saurabh, Douglass, S.A.: Net-centric act-r-based cognitive architecture with DEVS unified process. In: Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium. pp. 34–44. TMS-DEVS '11, Society for Computer Simulation International, San Diego, CA, USA (2011), http://dl.acm.org/citation.cfm?id=2048476.2048480
21. North, M.J., Howe, T.R., Collier, N.T., Vos, J.R.: A declarative model assembly infrastructure for verification and validation. In: Takahashi, S., Sallach, D., Rouchier, J. (eds.) Advancing Social Simulation: The First World Congress, pp. 129–140. Springer Japan, Tokyo (2007)

22. North, M.J., Collier, N.T., Ozik, J., Tatara, E.R., Macal, C.M., Bragen, M., Sydelko, P.: Complex adaptive systems modeling with repast simphony. Complex Adaptive Systems Modeling **1**(1), 3 (Mar 2013). https://doi.org/10.1186/2194-3206-1-3, https://doi.org/10.1186/2194-3206-1-3
23. Padgham, L., Scerri, D., Jayatilleke, G., Hickmott, S.: Integrating BDI reasoning into agent based modeling and simulation. In: Proceedings of the Winter Simulation Conference. pp. 345–356. WSC '11, Winter Simulation Conference (2011), http://dl.acm.org/citation.cfm?id=2431518.2431555
24. Rao, A.S., George, M.P.: BDI agents: From theory to practice. In: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95). pp. 312–319 (1995), http://www.agent.ai/doc/upload/200302/rao95.pdf
25. Rüb, I., Dunin-Kęplicz, B.: BDI model of connected and autonomous vehicles. In: 6th International Workshop on Engineering Multi-Agent Systems (EMAS 2018) (2018), http://emas2018.dibris.unige.it/images/papers/EMAS18-16.pdf
26. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, chap. 2.3.2. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edn. (2009)
27. Singh, D., Padgham, L., Logan, B.: Integrating BDI agents with agent-based simulation platforms. Autonomous Agents and Multi-Agent Systems **30**(6), 1050–1071 (Nov 2016). https://doi.org/10.1007/s10458-016-9332-x, https://doi.org/10.1007/s10458-016-9332-x
28. Tendeloo, Y.V., Vangheluwe, H.: An evaluation of devs simulation tools. SIMULATION **93**(2), 103–121 (2017). https://doi.org/10.1177/0037549716678330
29. Uhrmacher, A.M.: A system theoretic approach to constructing test beds for multi-agent systems. In: Sarjoughian, H.S., Cellier, F.E. (eds.) Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies, pp. 315–339. Springer New York, New York, NY (2001)
30. Uhrmacher, A.M., Kullick, B.G.: "plug and test" - software agents in virtual environments. In: 2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165). vol. 2, pp. 1722–1729 vol.2 (Dec 2000). https://doi.org/10.1109/WSC.2000.899162
31. Van Dyke Parunak, H., Nielsen, P., Brueckner, S., Alonso, R.: Hybrid multi-agent systems: Integrating swarming and bdi agents. In: Brueckner, S.A., Hassas, S., Jelasity, M., Yamins, D. (eds.) Engineering Self-Organising Systems. pp. 1–14. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
32. Varga, A., Hornig, R.: An overview of the omnet++ simulation environment. In: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops. pp. 60:1–60:10. Simutools '08, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium (2008), http://dl.acm.org/citation.cfm?id=1416222.1416290
33. Wainer, G.: Cd++: a toolkit to develop devs models. Software: Practice and Experience **32**(13), 1261–1306 (2002). https://doi.org/10.1002/spe.482
34. Wilensky, U.: Netlogo. http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL (1999), http://ccl.northwestern.edu/netlogo/
35. Zeigler, B.P., Praehofer, H., Kim, T.: Theory of Modelling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. Academic Press, San Diego, CA, USA, 2nd edn. (2000)
36. Zhang, Mingxin, Verbraeck, A.: A composable PRS-based agent meta-model for multi-agent simulation using the DEVS framework. In: Proceedings of the

2014 Symposium on Agent Directed Simulation. pp. 1:1–1:8. ADS '14, Society for Computer Simulation International, San Diego, CA, USA (2014), http://dl.acm.org/citation.cfm?id=2665049.2665050

Acknowledgement

We acknowledge the support of Cohort Systems, Ottawa, Ontario, Canada.

The work has been partially funded by Department of National Defence (DND) Contract Number: W7714-196749/001/SV.

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number 518212].

Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), [numéro de référence 518212].