Simulation-Aided Development of Multi-agent Algorithms for Tactical Missions

Antonín Komenda, Jiří Vokřínek, Michal Čáp, Michal Pechouček

Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague

Abstract—The development of intelligent control algorithms for robotic assets to be used in coalition operations is impeded by a significant gap between the way the theoretical artificial intelligence (A.I.) algorithms have been traditionally designed and validated and the way the practical algorithms for controlling robotic assets in (simulated) tactical missions are developed. On the first front, synthetic—usually mathematically defined—environments are used for the design and formal testing of A.I. algorithms, on the second front, low-level robotic simulators or even robotic field testing is employed.

This article introduces a development process and a simulation architecture that helped us to narrow this gap. We report on how we implemented such an architecture using a software toolkit for multi-agent prototyping called Alíte. The development process, the simulation architecture and the ready-made open-source toolkit Alíte allowed us to develop and integrate a number of intelligent control algorithms obtained by gradual adaptation of theoretical methods to more realistic environments.

Index Terms—Multi-agent Simulation, Tactical Mission, Coalition Operations, Algorithm Development

I. INTRODUCTION

In recent years, one could witness an intensified development and deployment of various robotic systems. One of the fastest-growing application domains for robotic systems are the Intelligence, Surveillance, Target Acquisition, and Reconnaissance (ISTAR) military missions performed by remotely controlled robotic assets. Often, the ISTAR missions form a part of a larger operation, where each coalition partner uses his own robotic assets. The individual robotic teams must be able to safely co-exist or even cooperate together. Currently, these robotic assets are controlled and coordinated exclusively by human operators. The scalability of such an approach is clearly constrained by the limits of human perception and the limits of inter-human interactions.

To address these constraints, there have been large investments in a successive introduction of autonomous multi-robotic systems. In such systems, the robotic assets use the techniques of distributed artificial intelligence to coordinate their actions and cooperate with each other. We will use the term Tactical Missions to denote the class of problems where multi-robotic teams carry out tasks in ISTAR missions, support disaster relief operations or assist humanitarian missions [9], [3].

The fundamental challenge associated with the multi-robotic application development is the deployment, validation and verification of the developed algorithms on real hardware. Conducting experiments on real-world robots is expensive both in terms of time and money. To reduce such costs, a simulation of the target system can be introduced. On the one hand, the experiments in a simulated world have the advantages of the reproducibility, direct control over the simulated world, and usually also the efficiency of experimenting (one can conduct batch experiments). On the other hand, the fundamental drawback of the simulated-world experiments is that the accuracy of the results depends on the fidelity of the world model employed by the simulation. Since the computational complexity of the simulation grows as a function of the realism of the underlying world model, high-fidelity simulations are often impossible to achieve due to their prohibitive computational complexity.

High-fidelity robotic simulators, such as Gazebo\textsuperscript{1} or Stage/Player\textsuperscript{2}, focus on precise simulation of physical dynamics of the robots. Such simulators can typically simulate only a low number of entities due to the computational complexity of the realistic simulation of robots’ dynamics. Further, they work on a single fixed level of simulation fidelity. On the other side of the spectrum, we have simulators used in serious gaming. An example is the Virtual Battlespace\textsuperscript{3} simulation or the Pogamut\textsuperscript{4} middleware. The first one targets human-in-the-loop military training scenarios, the other one serves as an integration middleware to various game engines that provides unified interface for experimentation with various A.I. algorithms.

The approach presented in this work stands in between of these two extremes. Additionally, it provides groundwork for integration of simulator components that work on different levels of simulation accuracy. The closest state-of-the-art simulator is MORSE\textsuperscript{5}. However, its focus is more on the robotic part (sensor, actuator and robotic base modeling), while we focus mainly on development, testing and validation of the target control algorithms with help of the simulation system.

\textsuperscript{1}http://gazebosim.org/
\textsuperscript{2}http://playerstage.sourceforge.net/
\textsuperscript{3}http://products.bisimulations.com/products/vbs2/overview
\textsuperscript{4}http://pogamut.cuni.cz/main/tiki-index.php
\textsuperscript{5}http://www.openrobots.org/wiki/morse/

\textsuperscript{1}This work was supported by US Army CERDEC grant no. W911NF-11-1-0252 and by the Grant Agency of the Czech Technical University in Prague grant no. OHK3-060/12.
II. Development Process

The presented development process is based on the simulation-aided design of multi-agent systems (SADMAS) methodology [8], [2]. The core principle of the SADMAS methodology is an iterative development process supported by approximated validation using testbeds of increasing fidelity. The goal of the process is a successful, cost-efficient deployment of the application on the target system, typically a hardware platform. The iterative process of the application development is based on the feedback from approximated testing. The extent of approximation can be described in two dimensions: level of abstraction (how much is the target system simplified) and scope of abstraction (which parts of the target system are simplified). In result, the initial system consisting of highly abstract algorithms is iteratively transformed, with increasing level of detail in each step, into a system deployable on the target hardware platform.

At the beginning, we employ the approach of theoretical A.I. and design the control algorithms in synthetic environments, described using general mathematical structures such as graphs and grids. However, right from the start, we perform the experiments on the algorithms within the framework of the target simulation system. This means that the interfaces between the control algorithm and the simulated environment must be flexible enough to allow easy redeployment of the algorithm to higher-fidelity simulation environments. After validating and verifying the algorithm in a synthetic environment, we can extend or replace parts of the simulation and re-validate the algorithm in a simulation containing more aspects of the target environment, i.e., having a lower level of abstraction.

Occasionally, after the abstraction of the simulation environment has been decreased, the tested algorithm has to be conservatively adapted. A conservative adaptation of an algorithm is an adaptation that preserves all the desired mathematical properties (e.g. soundness, completeness, etc.) for the price of possibly newly added domain-specific constraints on the validity of these properties. The final sum of such adaptations results in a theoretically-backed algorithm applicable in highly detailed simulated environments. The mathematical properties of the algorithm stay valid under the constraints introduced by the applied conservative adaptations.

A. Environment Model

Such a development process requires a simulator that offers a high flexibility in terms of scenario storyboards that can be constructed. In particular, one should be able to freely choose the simulation entities that form a simulation instance (this requirement is related mainly to the scope of abstraction in SADMAS) and the levels of detail on which are the entities simulated (this requirement is closely related to the level of abstraction in SADMAS) to enable successive adaptation of the tested algorithms.

A fundamental part of the simulation platform is a model of the virtual environment. To satisfy the above-stated requirements, we distinguish between the description of the simulated state and the state controllers, which animate the simulated world.

The state of the environment is represented by sets of state variable containers called state storages (see Figure 1). Each state storage is responsible for holding a specific part of the current state, i.e., all the state storages together constitute the full description of the current state of the simulated environment.

State controllers can be both a) the control algorithms tested in the simulated environment and b) program logic describing the mechanics of the simulated environment. The state controllers interact with the state of the environment indirectly through a set of interfaces called sensors and actuators. A sensor is an interface through which a particular part of the environment state can be read. Analogically, an actuator is an interface used to change a part of the environment state. Sensors and actuators are the only components that can directly access the state storages.

There are no a priori restrictions on what can a state controller model be, i.e., a controller can be a mechanism simulating physical laws of the environment (e.g. application of the gravity force to all simulated entities having mass), a simple reactive algorithm (e.g. simulation of swarm systems), or a complex deliberative algorithm (e.g. cognitive cooperating agents). The elements of the environment having no associated controllers remain fixed in their initial state. These can be e.g., the shape of the landscape, buildings, bridges, etc.

Furthermore, the sensors and actuators are not strictly limited to have a state controller on one side and a state storage on the other. A sensor or an actuator can be connected to other sensors or actuators, effectively forming an interface network. Such approach to the design of simulated environments leads to a significant cost reduction on implementation and debugging of the individual experimental scenarios, as the interface network can be flexibly reconfigured and the implementations of the individual sensors and actuators can be reused.

B. Simulation Process

The environment model has to be accompanied by a functional part, describing the behavior of the simulation. An
experimental validation in general requires statistical results from a large number of simulation runs. To ensure properties of the tested algorithms during the adaptation process, the simulation platform has to facilitate construction of experiment suites allowing execution of reproducible experiments.

While most of the abstract mathematical algorithms are well analyzed and strongly experimentally evaluated, it is much more challenging to design, run (and debug) replicable experiments in complex, high-fidelity robotic simulations involving dynamic entity behaviors and emergent behavioral phenomena. Large-scale simulations involve various aspects of non-determinism, which can lead to non-reproducible simulation runs. Such factors include parallel and random processes, as well as the limitations of the underlying hardware, such as CPU scheduling or memory swapping, etc. To ensure reproducibility of experimental runs, the simulator has to follow the concept of \textit{in vitro} simulation. That is a simulation that controls all the aspects of the modeled system. Besides controlling the evolution of the simulated world, the simulator must also have an ability to control the execution (i.e., suspend and later resume) of the validated control algorithms. Further, the simulator has to be immune to the race conditions and different results of process scheduling on the underlying computational infrastructure. Finally, any random processes involved in the simulation must be also under the control of the simulator, so that the same sequences of random events are generated in any two runs of the same experiment.

The need to execute large numbers of reproducible simulation runs turned out to hinge on the speed of simulation execution and the ability to make the runs deterministic on demand. To tackle this issue, departure from the classical exclusive model of centralized discrete time ticks and adoption of the \textit{event-based simulation mechanism} is required. This allows the system to disrespect real-time constraints of the wall-clock ticking mechanism and run the simulation as fast as possible given the available computational hardware resources (memory and CPU). The main advantage of this approach is that the time periods containing no simulation events can be skipped and thus the simulation runs significantly faster. However, at the same time the resulting simulator still features the ability to run at real-time simulation speed (for demonstration purposes or for hardware-in-the-loop experiment).

C. Example of a Multi-level and -scope Abstractions

The model based on state storages, sensors, actuators, and loosely coupled controllers offers high-levels of flexibility. In result, a programmer can add and remove new types of simulated entities easily (scope of abstraction) and easily switch between different types of simulation modes (level of abstraction) for individual entities. Moreover, as expected from a multi-agent simulation, the model also offers scalability in terms of numbers of simulated entities.

For instance (see Figure 2), we can define three types of abstractions for unmanned ground vehicles (UGVs) used in the simulation and represent them by three separate state storages \texttt{DiscreteUGVStorage}, \texttt{InterpolatedUGVStorage}, and \texttt{PhysicalUGVStorage}. The first one describes the current state of a UGV by a node on a street graph. The second one enriches the by-node description by a position vector $(x, y)$ representing the position of a UGV on a 3D mesh of the ground surface. The last abstraction extends the state further with a description of a fully dynamic state comprising position $(x, y, z)$, velocity $(\dot{x}, \dot{y}, \dot{z})$, acceleration $(\ddot{x}, \ddot{y}, \ddot{z})$ and rotational components $(\phi, \theta, \psi), (\dot{\phi}, \dot{\theta}, \dot{\psi}), (\ddot{\phi}, \ddot{\theta}, \ddot{\psi})$. To control the state stored in these storages, we use three actuators \texttt{GoToNodeActuator}, \texttt{MoveInDirectionActuator}, and \texttt{SteerAndAccelerateActuator}. One can implement an actuator to control the respective state storage directly, but it is also possible to implement an actuator to control storages indirectly through other actuators. In practice, such coupling will result in an algorithm that recursively translates the higher-level control to lower-level control. For example, if a UGV state controller based on the node-to-node mode of navigation is required to drive a physically simulated UGV, it uses the following actuator sequence: \texttt{GoToNodeActuator} $\rightarrow$ \texttt{MoveInDirectionActuator} $\rightarrow$ \texttt{SteerAndAccelerateActuator}. As we can see now, we can use any of the presented state storages as long as the controlling algorithm uses only the top-most actuator, i.e., \texttt{GoToNodeActuator}. In effect, we can design a high-level algorithm controlling a UGV only on node-to-node basis using \texttt{GoToNodeActuator}, but we can immediately test it in all prepared levels of abstraction (discrete, interpolated, physical).

To extend the scope of the simulation, we add simulated vertical take-off and landing unmanned aerial vehicles (VTOL UAVs). Similarly to the simulated UGVs, VTOLs have three levels of abstraction represented by three state storages and three related actuators. The actuator sequence follows the same pattern as in UGVs.

The last example enriches the simulated environment with entities representing troops. Here, we create only two levels of abstraction represented by two state storages \texttt{DiscreteTroopStorage} and \texttt{DirectedTroopStorage}. The first level of abstraction is similar to \texttt{DiscreteCarStorage} (representing the position of a trooper in terms of street graph nodes), the latter describes the ground position and the heading angle $(x, y, \phi)$ of a trooper. We create a \texttt{WalkToNodeActuator} and \texttt{MoveAndTurnActuator}. We cannot reuse \texttt{GoToNodeActuator} in place of \texttt{WalkToNodeActuator} since the UGV actuator uses a different control logic to simulate the movement (although the input parameters and the results are identical for both the actuators—both the UGVs and the troops move from one node to another—for the UGV, the duration of the movement can be computed from the engine power, for the trooper the duration of the movement can be, for instance, a function of the weight of the personal gear carried).

From this point, we have separate components for a UGV, a VTOL and a trooper in the model of environment. In a simulation run, we can use these components separately (just UGVs or just troops) or we can mix them together (e.g. troops following a car). Moreover, we can mix different levels of abstraction of various components (for instance, a transportation UGV using trajectory interpolation representing...
a convoy is followed by physically simulated vehicles representing UGVs accompanied by troops having position and direction representing the support squad protecting the convoy against adversaries moving on node-to-node basis blocking junctions on the street map).

III. MULTI-AGENT TOOLKIT ALITE

The development process together with the requirements on the simulator led to the architecture described in the previous section. The strong emphasis on the flexibility of the interface between the controllers and the simulated environment requires equally flexible software tools able to help with the implementation of such a simulation system. Alite is a software toolkit that provides such support out of the box.

Alite is a software toolkit simplifying implementation and construction of (not only) multi-agent simulations and multi-agent systems. It stands on technologies related to JVM ecosystem and it is mostly written in Java. The objectives of the toolkit are to provide a highly modular, flexible and open set of functionalities supporting rapid prototyping and fast implementation of multi-agent applications, mainly focusing on highly scalable and complex simulated environments. The guiding principles underlying the Alite design are i) modularity, so that the system does not commit a developer to a specific definition of concepts such as agent, environment, etc. and ii) composability, so that the various components of the toolkit can be put together in a rapid and flexible manner. In result, Alite can be seen as a collection of highly refined functional elements providing clear and simple APIs, allowing a programmer to put together relatively complex multi-agent simulation scenarios rapidly. In the following paragraphs, we explain the main characteristics and distinguishing features of Alite.

Alite addresses the problem of multi-agent platform resilience in the face of the need to incorporate various a priori unknown future requirements by variability in composition of functional elements. The number of possible combinations allows for construction of a wide spectrum of structurally different multi-agent applications. This feature distinguishes Alite from the pre-designed frameworks such as Jade, Cougaar and Aglobe multi-agent platforms. As multi-agent application’s requirements evolve, the requirements on the agent platform itself are changing. Alite does not provide a single platform for all, but rather offers an efficient way to build a platform that fits the specific needs of the multi-agent application under development. The application can make use of one or more functional elements available in Alite toolkit.

Among others, Alite provides an implementation of building blocks introduced in Section II-A. An application developer can put together different parts from Alite toolkit to implement a multi-agent simulation platform that targets specific requirements of the application in question. In particular, it is designed to facilitate implementation of simulations that adopt the in vitro principle and the event-based simulation mechanism as described in Section II-B. Further, Alite contains functional blocks supporting (i) inter-agent communication, (ii) configuration and initialization, and (iii) visualization. The communication package provides an easy-to-use interface that can be integrated with a number of message passing channels. For the tactical mission simulator, we use a message passing mechanism implemented using the event-based simulation. The configuration and initialization uses the dynamic programming language Groovy to configure the parameters and initialize the initial state of the simulation in a concise and flexible manner. Such a flexibility allows a programmer to experiment with structurally different simulation scenarios, a must-have for a successful adoption of the SADMAS approach. Finally, the state of the simulated world can be displayed using the 2D/3D visualization component. This component is designed as fully separable from the simulation core, therefore if the visualization component is turned off, there is no efficiency burden or negative influence caused. We released the entire tactical mission simulator as a standalone package under an open source license.

The power of Alite’s loosely coupled design has shown its benefits during the construction of a multi-agent simulator of distributed tactical missions described in this work. A number

---

8http://jade.tilab.com/
9http://www.cougaar.org/
10http://agents.felk.cvut.cz/aglobe/

---

Figure 2. State storages and related actuators for description of three levels of abstraction for UGVs, three levels of abstraction for VTOL UAVs and two levels for simulated troops. The scope of abstraction is demonstrated using three different types of simulation entities, which can be variably used together. There is one Interface block and one Environment block divided into three views from perspectives of the particular simulated entities.
of tailor-made domain-specific components integrated with the Alite infrastructure enabled us to transparently combine and validate several A.I. algorithms and control programs written in agent-oriented programming language Jazzyk [7] in a complex environment simulation comprising realistic physical simulation of rigid-body models based on JBullet simulator\textsuperscript{12}. Another Alite-backed multi-agent application of physically simulated UGVs for the domain of multi-agent cooperation and coordination in complex urban environments has been presented in [10].

IV. ALGORITHMS AND VALIDATION

To demonstrate the key ideas of the proposed approach, we present an example domain. As we can see in Figure 3, the simulated environment comprises a village, the surrounding uneven landscape, models of the available robotic assets, friendly persons and adversarial persons. We consider two main groups of robotic assets: UGVs and UAVs. Additionally, on all levels of fidelity, we distinguish Conventional Take-off and Landing (CTOL) UAVs and VTOL UAVs as their movement models fundamentally differ. To enrich the set of possible tasks for the robotic assets, we also simulate the behavior of friendly (blue) and enemy (red) forces forming teams and convoys.

The proposed development process has been used during design and implementation of three A.I. algorithms that have been employed in an evacuation tactical mission. The mission consist of combined ground, aerial and human teams operating in an urban area. The algorithms support the operations such as protection of ground teams (possibly of various coalition partners) by an UAV, pursuit of a smart target (using heterogeneous airborne and ground assets), or multi-asset plan repair due to the dynamic environment. The application of each of the algorithms consisted of (i) analytical design based on the state-of-the-art techniques, (ii) implementation and validation using a discrete synthetic environment then adapting it to (iii) a dynamic continuous environment and finally adjusting it to (iv) complex mission environment and validation using the integrated tactical scenario.

Table I shows the list of operations supported by application of scientific algorithms and corresponding levels of abstractions. Overview of the scientific methods applied on specific mission operation problems is summarized in Table II together with main development problems introduced by changing the levels of abstraction. Short description of the validated mission operations follows, more information can be found in [4]. Similar approach to the one presented here was also used in the domain of dynamic route planning and cooperative navigation in a robotic team [10].

**Ground teams protection** against the attacks from the adversaries is one of the parts of the decentralized tactical mission. The protection is carried out by a small team of aerial vehicles, which need to randomize their movement strategy to minimize the chance of the team being attacked in the worst timepoint. The solution is based on optimal strategies for patrolling game in urban environment [1]. All the adaptations of this scientific method towards environment models having higher fidelity, were only a matter of slight adjustment of the algorithm implementation and posed no crucial problems. The most likely explanations behind the straightforward development process is the low computational complexity of the algorithm (the set of applicable strategies was precomputed) and a intrinsic temporal flexibility in execution of such strategies.

**Smart targets pursuing** is another example operation in the mission. A smart target (i.e., a target, who actively monitors its surroundings and acts accordingly) actively tries to avoid the tracking unit. In this case a formal game-theoretical model of a pursuit-evasion scenario with heterogeneous teams of agents and resulting optimal algorithm is used [6]. The adaptation of the algorithm among the different abstraction levels also turned out to be straightforward. This is due to the any-time property of the experimented algorithm.

**Multi-agent plan repair** is needed to plan activities for a team of agents in the highly dynamic environment. The solution is based on a multi-agent planner and plan repairing algorithms preserving parts of the old plans and therefore minimizing communication among the participating agents [5]. The adaptation of the algorithm for the domain of tactical support led to an introduction of a restricting condition on the depth of the search tree to limit the computational complexity of the search. The plan-repairing mechanism also addresses the problems caused by the uncertain movement of the troops in the dynamic environment.

V. FINAL REMARKS

One can come up with various approaches to the development of multi-agent algorithms for missions in context of coalition operations. However, according to our experience such problems are typically so complex, that the first-shot approaches usually fail. We are providing a description of a well-tried approach based on SADMAS methodology that specifically focuses on the domain of tactical missions in dynamic environments.

\textsuperscript{12}For more information on the physics simulation see JBullet (http://jbullet.advel.cz/) – a Java port of Bullet Physics Library (http://bulletphysics.org)
We provide a general description of the approach, so that any available software solution that is suitable for the problem can be used for implementation. Moreover, we give an overview of a freely available software toolkit Alite and report on how it can be used to implement a simulation system that supports the presented development approach.

Finally, we conclude the work with an example multi-agent application that was developed with the help of the presented techniques. It employs game-theoretic, plan repair and multi-agent coordination algorithms. The application demonstrates the use of such algorithms to control a simulated robotic team that supports simulated troops in an evacuation mission.

The most promising directions for future work is a design of well grounded processes and software tools that would support automated, or at least semi-automated, design of the multi-level and multi-scope abstractions.

## References


