Simulated Multi-robot Tactical Missions in Urban Warfare

Peter Novák, Antonín Komenda, Michal Čáp, Jiří Vokřínek and Michal Pěchouček

1 Multi-robotics in urban warfare

Since late 90's of the last century, rapid advances in technology, mechanical engineering, miniaturization, telecommunications and informatics enabled development and routine deployment of sophisticated robots in many real world domains. Besides many applications in assembly industry, e.g., in car, or electronics assembly lines, defense organizations, together with space exploration and mining industries belong to the most demanding and optimistic users of robotic technology [23]. Especially in the military domain we nowadays witness a routine deployment of robotic assets in the field. Some of the popular examples of such robots include unmanned aerial vehicles/systems (UAV/UAS), be it conventional fixed-wing aircrafts (CTOL - a conventional take-off and landing vehicle), various rotorcrafts, such as single, or multi-rotor helicopters (VTOL - a vertical take-off and landing vehicle), autonomous underwater vehicles (AUV), unmanned cars (UGV - a unmanned ground vehicle), or unattended ground sensors (UGS), such as various acoustic, seismic and chemical sensors, or cameras. Various size and equipment classes of such robots are used in tactical, law enforcement or rescue operations for tasks, such as security surveillance of urban areas, firefighting, or providing situational awareness, mapping and exploration of areas stricken by natural disasters, or tasks in dangerous work environments, such as stabilization of damaged nuclear reactors after an earthquake. The robots are usually performing either manipulation tasks, or provide situational awareness to human task forces by information collection, such as continuous video streaming, acquiring static imagery or analysis of chemical, or nuclear hazards [17].

Even though we recently witnessed rapid advances in control of robots in scenarios such as e.g., autonomous cars [24], or service robotics [32, 6], the state of the

Peter Novák, Antonín Komenda, Michal Čáp, Jiří Vokřínek and Michal Pěchouček
Agent Technology Center, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Technická 2, CZ-16627 Prague 6 - Dejvice, Czech Republic, e-mail: `{peter.novak,antonin.komenda,michal.cap,jiri.vokrinek,michal.pechoucek}@agents.felk.cvut.cz`

art in high-level control of robotic assets still relies mainly on teleoperation. Some of the high-profile examples of such deployed systems used by military and law enforcement are the surveillance drones providing imagery and video streams from operation theaters [30], or robots dealing with relief operations in damaged nuclear power plants [15].

While teleoperation is a very effective method of robot control in many scenarios, it does not scale well with the increasing number of deployed assets. One of the most important problems arising in situations where a larger number of cooperating robots is needed to successfully accomplish a joint mission are the limits of the employed resources. A single operator is capable to directly control only a relatively small number of robots. Thus with increasing number of deployed assets, direct control of robots by humans becomes too costly in terms of human resources. In result, scaling the number of robots requires larger numbers of human controllers, what finally raises also financial and logistical costs of such operations.

One of the natural solutions to the problem of high-level control of multi-robot teams is a significant increase of the level of autonomy of the individual robots, as well as the multi-robot team itself. I.e., instead of resting on a human operator, the tedious lower-level decision making and control (e.g., movement in a terrain, or camera pointing, simple task planning, etc.) should be shifted on board to the robot itself and the human operator should only take care of tasking the multi-robot team and oversee the mission execution. The working hypothesis underlying the solution is that

> *a single operator is capable to task and oversee even a relatively large multi-agent teams comprising of highly autonomous robots which require human intervention only rarely, especially when facing crucial choices in execution of the joint team mission.*

Agent Technology Center (ATG) at the *Department of Cybernetics* of the *Czech Technical University in Prague* is one of the leading research groups in innovative industrial and research applications of multi-agent systems. Besides other research topics, one of the major realms of its activities is research and proof-of-concept prototyping in the field of mid and large scale simulations of multi-robot systems. In this area, ATG focuses primarily on teams of autonomous aircrafts and ground vehicles. In this chapter we present a cluster of completed projects Tactical AgentFly and Tactical AgentScout, as well as outline the core objectives of an on-going follow-up project AgentFly-In-Air. All these projects aimed at investigation of cooperation and coordination issues in multi-robot teams either carrying out tactical missions in urban warfare scenarios, or providing information collection support to human troops on the ground in such environments. The particular objectives and foci of interest of the projects were organically evolving over time spanning the years 2008–2011. However, the specifications of the individual workpackages and their respective delivered demonstrators provide a set of high-level design requirements on an underlying technological infrastructure.

The main contribution of this chapter is an account of architectural and technological issues related to development of the multi-agent platform and simulation subsystems for the project cluster. The underlying storyline revolves around the architectural shifts during the project development caused by gradual extensions of the technology, as well as incorporation of often conflicting application requirements over time. In essence, these can be characterized as a move from a general-purpose MAS platform imposing a specific MAS philosophy towards a more liberal component-based architecture in terms of a toolkit for rapid construction of application-specific fragmentary MAS platforms and applications.

In this respect, firstly, Section 2 provides an overview of the research objectives of the twin projects. Subsequently Section 3 discusses the initial analysis of the approach to design of the underlying multi-agent technological infrastructure. Section 4 gives a more detailed account of the completed projects, the approaches we took to tackle them, as well as the evolution of the underlying technological infrastructure and the involved toolkits. Critical analysis of the technological infrastructure evolution provided in Section 5 highlights the main lessons we learned in the course of the work on the twin projects. Finally, Section 6 concludes the chapter by an outlook to the open issues and challenges for the broader MAS community involved in development and testing of various decentralized algorithms ultimately targeted for multi-robot systems embedded in real hardware.

Sections 3, 4 and 5, the three core sections of the presented chapter are all structured in a similar manner and subsequently discuss the main aspects of the implemented system. Namely the issues of simulation and environment modeling, followed by tackling the problems involved in experimental evaluation of the research algorithms under investigation, together with configuration of simulation scenarios. Subsequently, we discuss topics in mechanisms for agent deliberation and behavior implementation and conclude by treatment of the issues in simulation visualization and user interfaces. The recurrent structure provides scaffolding for the main storyline of the chapter, the evolution of various aspects of the technological platform underlying the Tactical AgentFly and Tactical AgentScout project cluster over time.

2 The project cluster: Tactical AgentFly, Tactical AgentScout

In the course of the years 2008–2011, Agent Technology Center was (and still is) involved in a continuous interaction with CERDEC, ONR and AFOSR, the research and development departments of U.S. Army, Navy and Air Force. Over time, these interactions resulted in formulation of several research problems stemming from the real needs of military units carrying out tactical missions on the ground and their usage of advanced robotic and sensory technologies, such as various aerial drones, especially the class of man-portable small unmanned fixed-wing aircrafts, various rotorcrafts and unmanned ground vehicles. In particular, some of the most prominent problem topics included area exploration, surveillance, tracking of mobile targets, patrolling and teamwork coordination in structured heterogeneous multi-agent teams. The work towards investigating these research issues produced a number of technological challenges, which had to be solved and implemented using an under-

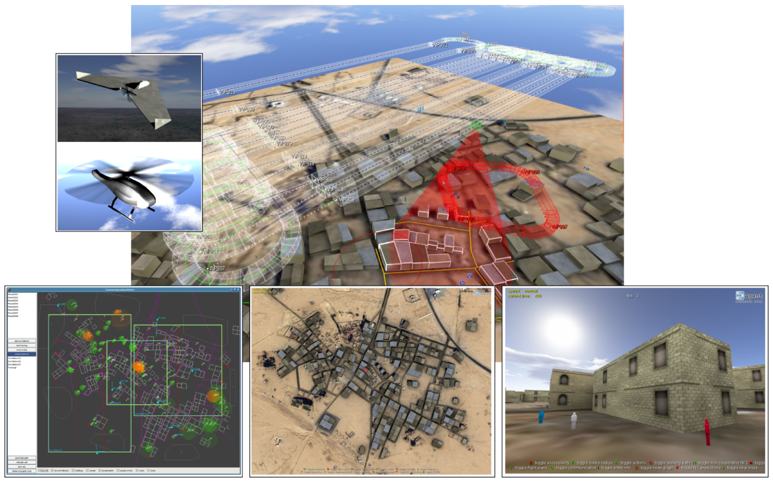

Fig. 1 Visual impression of Tactical AgentFly simulated environment

lying technological infrastructure providing the basis for development of prototypes demonstrating the proposed research approaches.

From technological perspective, the common character of the here described projects is that the deliverables take the form of executable demonstrators showcasing behaviors of algorithms for coordination of teams of robotic aircrafts, ground vehicles and human troops in synthetic simulated scenarios of tactical urban warfare. Besides the capability to empirically and reproducibly evaluate the performance of the developed algorithms, the simulations must come with rich visualization components. One one hand, rich visualizations plausibly demonstrate that the behavior of the robots complies with their realistic limitations, such as sizes, weights, speeds, physical motion dynamics, and physical properties of the environment. On the other, they also help to facilitate dissemination of the results beyond the particular sponsoring partner organization to non-expert audiences. In a consequence, the form of the projects' deliverables frames the more detailed technological requirements stemming from the research objectives of the projects. In this section we provide an overview of the research issues of the individual phases of the projects Tactical AgentFly and Tactical AgentScout.

2.1 Tactical AgentFly

The objective of the TACTICAL AGENTFLY project was to develop basic agent-based techniques for controlling a group of autonomous UAVs performing information collection in support of tactical missions. The emphasis was on accurate modeling of selected key aspects occurring in real-world information collection tasks, in particular physical constraints on UAV trajectories, limited sensor range and sensor occlusions occurring in spatially-complex environments. The ultimate goal was to provide a high-level interface through which the operator can control a fleet of UAVs and assign high-level tasks to the multi-robot team. The task allocation to individuals UAVs itself, as well as planning of their optimum trajectories was performed automatically by the multi-agent team. The specific objectives of the project were the following:

Formal framework: we had to design a framework for formal specification of the information collection problem, serving as a common reference point for the rest of the project. In particular, we were asked to investigate the concept of an information collection task, including task constraints and task objective functions, which formed a basis for evaluation of the developed algorithms.

Persistent area surveillance: we investigated mechanisms for control of operation of teams of UAVs providing and maintaining an up-to-date operational picture of a designated target area. A key feature of the developed surveillance algorithms is their respect for UAV's motion constraints and the ability to provide full area coverage even in environments affected by sensor occlusions, such as narrow streets between tall buildings. The results of this research track have been published in [28].

Target tracking: we were exploring the mechanisms to control the operation of an UAV providing continuous tracking of one or multiple mobile ground targets, respecting the UAV's motion constraints.

Information collection testbed: an important objective was to develop an extensible software platform for implementing, simulating and evaluating various UAV control mechanisms explored in the project. It had to contain a detailed model of the urban environment, a model of the UAV's on-board camera and a behavioral simulation of several types of ground entities. A simulation of a multi-stage search-and-capture mission was to be prospectively implemented as well in order to enable evaluation of information collection mechanisms on a real-world-like scenario. The testbed had to provide intuitive real-world-like 3D visual output allowing the presentation of the project results outside the strictly technical community.

Command and control (C2) user interface: we had to develop an integrated C2 system for mixed-task information collection. Such a system was to provide an additional level of automation on top of the autonomous surveillance and tracking control mechanisms. It should have consisted of two parts. Firstly, a C2 panel through which the operator can specify information collection tasks and inspect their results, and secondly, an allocation algorithm which optimally allocates a mix of concurrent information collection tasks between a group of UAVs.

After the project's first phase was completed, we were awarded a follow-up project, during which we moved from the basic coordination algorithms for teams of fixed-wing UAVs towards more advanced techniques applied in heterogeneous teams. The main research objectives were the following:

Modeling Vertical Take-off and Landing (VTOL) Assets: we had to extend the existing platform and enable integration of various types of VTOL UAV assets (helicopters, quadrotors, etc.). The second major research objective of the project was to propose and develop suitable algorithms for trajectory planning of VTOL assets and integrate them with the VTOL model. The resulting algorithms have been presented in [10, 11].

M×N tracking: we focused on investigation of algorithms for tracking larger numbers of mobile targets by relatively small teams of aerial assets. One of the aims was to investigate the methods of maximizing persistence of tracking of the objects and identifying how many assets are needed in different types of tracking scenarios. Concretely, this research led to investigation of techniques for intelligent target tracking task hand-over between multiple UAVs. The results of this research track are discussed in [31].

Coordination for mixed information collection activities: in this workpackage, we aimed at studying mutual interactions between simultaneously performed heterogeneous information collection tasks. For this we had to i) extend the range of considered information collection task types with additional classes, in particular exploration and search, and ii) study the theoretical interactions between mixed/heterogeneous information collection tasks performed simultaneously. In particular, the idea was to investigate the problems arising from automated techniques facilitating transparent switching between different information collection tasks, such as switch from surveillance to target tracking and back.

Mission-centric/oriented information collection: the final objective of the second phase of the Tactical AgentFly project was to further extend the integrated coordination for mixed information collection. That is, the task was to propose techniques for a team of UAVs performing information collection tasks, however taking into account the plans of special operation units carrying out their own mission on the ground in the town, so that the information needs of the mission are optimally covered. The activity aimed at research and possibly prototyping efforts towards considering temporal development and dependencies between the individual information collection tasks as the mission progresses.

2.2 Tactical AgentScout

TACTICAL AGENTSCOUT project started as a branch of TACTICAL AGENTFLY project that aimed at integration of aerial information collection with various types of ground robotic assets. The foci of the project were on multi-agent planning and task-allocation problems. The proposed solutions were to be demonstrated in a simulated tactical scenario taking place in a complex urban environment defined by an *a priori* known street map. Furthermore, the environment included various 3D terrain features, as well as buildings. Finally, the the environment should contain a number of road blocks on the street map, which however were not *a priori* known to the multi-robot team. The goal was to find a safe and effective path for a convoy to traverse the urban environment. The task of the team of autonomous UGVs was to cooperatively support the convoy by continuous exploration of the area and search

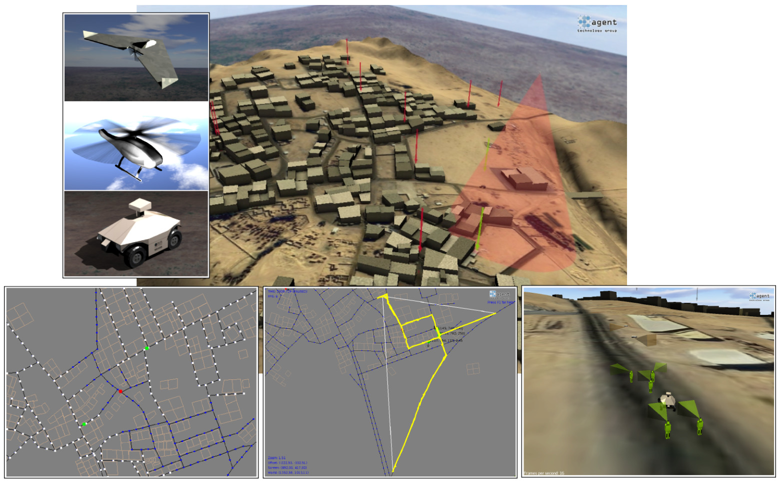

Fig. 2 Visual impression of Tactical AgentScout simulated environment

for the road blocks on the way of the convoy. The information about the discovered obstacles is then communicated to the convoy and can be used to update its plan. In particular, the project included the following research topics:

Integration of various ground units: the main planned contribution of the Tactical AgentScout project was to enable integration of various autonomous ground robots into the existing platform. In particular, the challenge was to plausibly model the physical dynamics of the assets within the simulation.

Planning **under uncertainty:** the flight trajectory planning algorithms for UAVs as used in Tactical AgentFly assumed reliable execution of the generated plans. In the domain of ground vehicles, we deemed such an assumption too strong. The goal was to develop a path planner for UGVs that is able to efficiently control the vehicle also in dynamic, uncertain environments. To evaluate such a planner, we aimed to simulate the UGVs using physically realistic models that let us generate some of the problematic real-world phenomena such as tire spin, vehicle mass momentum, limited engine power etc.

In a follow-up project we moved towards more advanced techniques studying various aspects of adversarial and cooperative behaviors in dynamic environments. Specifically, the project addressed the following research challenges:

Patrolling of mobile targets: a complementary task to tracking a number of mobile adversary targets is the protection of mobile ground units against attacks from enemy units. The motivating scenario is an urban environment with a number of convoys passing through an area which should be protected by a small mixed team of aircrafts, e.g., helicopters and small fixed wing aircrafts. In such a scenario, it is vital for the patrol to execute a non-predictable patrolling strategy. The opposite would allow the adversary to optimize with respect to the patrol's strategy and attack the convoy in the worst timepoint, e.g., when the patrol just left the convoy it protects. The solution was use of randomized strategies which, however, still maintain certain average frequency of visits of each protected convoy. The research objective was to develop algorithms for computation of optimal strategies for protecting of mobile targets in adversarial environments. The resulting method has been published in [7].

Smart targets modelling: the mobile target tracking techniques as proposed in the Tactical AgentFly project paid no attention to intelligence of the tracked targets. As an extension, in this project we considered smart targets, i.e., such which actively monitor their environment and optimize their behavior to act with respect to the information they acquire. Smart targets, when being tracked, are aware of that fact and actively try to avoid the tracking unit. Complementary, we consider trackers (be it UAVs, VTOLs, UGVs, or personnel) to be aware of the fact that the tracked targets are aware of their activities and try to act in best response to the whole setup. The objective here was to propose and solve a formal game-theoretical model of pursuit-evasion scenario with heterogeneous teams of agents.

Multi-agent re-planning and plan repair: classical-style planning is not robust with respect to unexpected events occurring in dynamic environments. The standard solution, in such cases, is to simply re-plan the agent's behavior from scratch. While decentralized extensions of classical planning can be used to compute activities of the individual team members, full-scale re-planning can become too costly due to the costs of communication in multi-agent teams executing a decentralized planning algorithm. Thus, the objective was to formalize multi-agent plan repair problem and devise and evaluate algorithms for solving it. An initial version of the plan repairing algorithm has been presented in [22].

Coordination and teamwork: reactive planning is an alternative approach to dealing with dynamics of environments. The state-of-the-art techniques of reactive planning, however, do not support implementation of team-level behaviors. The objective here was to extend an existing agent programming framework so that it can accommodate techniques for team-level coordination specification in terms of reactive plans executed jointly by the team members. In particular, we aimed at implementation of the *distributed commitment machines* approach [33] in a chosen agent-oriented programming language.

3 Analysis and design of the system

The general description of the individual research issues discussed in the previous section and their respective software demonstrators provides a comprehensive overview of the various, often conflicting, requirements on the technological infrastructure for the Tactical AgentFly and Tactical AgentScout project cluster. The overall goal of the technological side of the whole endeavor, the main topic of this chapter, could be formulated as follows:

> *Develop a set of software tools enabling rapid prototyping of a broad range of simulated missions involving teams of autonomous robotic assets, as well as various situation scenarios in tactical urban warfare the robots are involved in. Furthermore, the toolkit must allow for batch evaluation of performance of the algorithms governing the behavior of the multi-agent teams in a range of configurations of the simulated scenarios.*

In the core, the objective was to facilitate modeling and execution of various types of robotic systems which i) feature heterogeneous physical capabilities, ii) employ heterogeneous control algorithms governing their autonomy, and finally iii) are embodied in simulated, but realistic physical environments. As already discussed in the introductory section, the underlying hypothesis of the research work in the twin projects was that a significant increase of autonomy of the individual robots and the multi-robot team itself will lead to reduction of human resources involved in the operation and monitoring of such teams. The stress on the autonomy of the involved agents, such as various unmanned aerial or ground vehicles, directly induces the remaining requirements usually ascribed to *intelligent agents*[34]. In particular these include reactivity, proactivity and in our context especially social capabilities. In result, the character of the problem directly correlates with the usual assumptions underlying the multi-agent paradigm. Hence the choice of the conceptual and analytical framework for multi-agent systems, as well as the initial tools and platforms for development of multi-agent systems became natural and straightforward. Additionally, our hypothesis was that the inherent decoupling of entities according to the multi-agent systems paradigm should open a way towards future porting and deployment of the developed agent (simulated robot) control algorithms to real hardware. We also hypothesized that the decoupling, i.e., the inherent modularity of multi-agent systems will facilitate rapid prototyping of a broad range of simulated scenarios involving heterogeneous multi-robot teams. It also should have have options opened for rich modeling of environments these robots operate in. Finally, the previous industrial development and deployment of multi-agent systems, as also discussed in [3], provided ample argument in favor of future scalability of the developed multi-robot simulations to relatively large numbers of involved agents.

The remainder of this section provides a discussion of the initial analysis and design consideration tackling the technological problem stated above. Subsequently, we provide an overview of the technologies and platforms we initially chose for implementation of the first phases of the project cluster. In Section 4 below we discuss our experience with these technologies and how the evolution of the projects over time led to reconsideration of our initial choices.

3.1 Initial system requirements

During the initial system analysis step of the first phase of the Tactical AgentFly project, we identified a list of architectural and functional requirements on a technological infrastructure underlying the project implementation. These can be divided into four groups subsequently discussed below. The structuring along the four aspects of the technological infrastructure also provides a scaffolding for discussion of the system evolution in time and its critical analysis in the subsequent parts of the chapter.

3.1.1 Multi-agent platform

Given the argument in favor of application of multi-agent paradigm in the projects and the stress on future potential for porting and deployment of the developed algorithms to real hardware assets, there arises a need for a multi-agent platform. I.e., a software supporting modeling of the individual agents, execution and life-cycle management of the multi-agent system as a whole and services of a communication middleware, such as a white pages register and a discovery service. Due to the focus on relatively small and mid-scale simulation scenarios involving dozens, possibly low hundreds of agents and environment entities, the requirements on network and CPU distribution and agent mobility agent mobility were not an important issues in our projects. Finally, since organizational structuring and coordination of MAS teams was the main research focus on the application level in the projects cluster, we felt that binding to a particular MAS organization philosophy would be rather a burden possibly interacting with the coordination techniques under investigation. Thus, an *a priori* organizational model imposed by the MAS platform was undesirable.

3.1.2 Environment simulator and scenario modeling

A simulated multi-robot system must be embodied in a physical environment which faithfully models a set of relevant aspects of the real physical reality in the simulation. Thus, the technological platform should facilitate rapid development and configuration of instances of simulated environments. First and foremost, such scenarios comprise the physical structure and topology of the environment. In particular, our focus on tactical military operations in urban terrain dictated ability to model landscapes with realistic terrain, traffic infrastructure of small and medium urban areas including buildings, roads and bridges. Finally, due to the focus on robots physically interacting with the environment, such as unmanned ground vehicles, we needed to realistically model their physical interaction with the environment including phenomena of gravity, object masses and rigid body interactions. Due to the need to demonstrate functionality and robustness of the developed coordination mechanisms in various settings, the platform should provide means for straightforward configuration of the physical features of simulated environments in different simulation instances.

The platform should also support clear cut interfaces between agents and the environment. That is, the software interfaces for implementation of robots' sensors and actuators must be well defined so that i) the agents use unified style of such interfaces across the whole application, and ii) in the case of future need, these could be easily bridged to physical hardware sensors and actuators.

3.1.3 Experiments and configuration

The main objective of the described projects is to investigate various types of coordination mechanisms among agents belonging to teams of cooperative individuals. Thus, besides proposing and developing such algorithms, the main task on the application level is to perform empirical evaluation of their performance and measure it against benchmark cases and alternative state-of-the-art methods. Running such experiments *en masse* requires that the platform is capable of providing means for batch execution of experimental setups in different configurations and enables to collect the resulting data sets in a straightforward manner. Furthermore, to facilitate reproducibility of the experiments, the experiments should be run in a deterministic manner with pre-configured random generators if needed. Finally, since there might be hundreds of such experiment scenario instances, the platform must be able to execute them in parallel, as well as in faster than real-time speed. I.e., it must support both modes of scenario execution: physical-time-speed mode, where one objective second correlates with one second in the simulation; and it should be able to run at the top use of the available resources, such as the CPU speed, operating memory limit, etc.

3.1.4 Agent behavior control

On one hand, often in the project cluster experiment setups the individual behavior of the simulated robots was rather rich beyond the particular coordination mechanism under investigation. For instance, while an autonomous ground vehicle must coordinate its plans for traversal of a given town street network with its peers, at the same time it is responsible for local execution of those plans. Besides steering the physical body of the robot and monitoring the progress of the plan execution, this task also involves relatively complex behaviors for dealing with the issues which were abstracted away by the planner. For instance, the plan would prescribe driving through a sharp curve in a narrow street, but due to the physical limits of the vehicle the car could end up stuck in the corner and must maneuver out of the place e.g., by iteratively driving backwards and forwards.

On the other hand, the environment contains also actors, which are not in the focal point of the multi-agent coordination mechanisms. These include agents representing the ground troops carrying out a tactical operation (the blue force), which the robots are supposed to support. Furthermore, there can be also various kinds of adversarial units (the red force), or civilians. Depending on the particular scenario, their behaviors range from relatively simple, such as random movement within a perimeter, to relatively complex ones implementing the mission played by the blue force. From the point of view of the multi-robot team in consideration, such agents model, as well as generate the environment dynamics relevant to the target coordination mechanism under investigation.

Finally, while some simulations would require rather abstract discrete time stepwise simulation of agent's actions, in other scenarios more realistic fully asynchronous time model is needed. Due to uncertainties with respect to the particular style of the simulations required for evaluation of the broad range of the considered centralized and decentralized coordination algorithms, the simulator shouldn't be strongly bound to a single simulation model.

3.1.5 User interface and visualization

To maximize applicability of the algorithms on real hardware robots in the future, one of the requirements of the projects Tactical AgentFly and Tactical AgentScout was that the developed coordination mechanisms must be demonstrated in scenarios featuring believable adversaries, and close-to-real-world physical environments. In result, the underlying technological infrastructure had to support rich visualization interfaces providing both 2D and 3D graphical views on the state of the simulation capable to display the physical dynamics of the simulated entities in real-time. Finally, the platform had to support creation of graphical user interfaces for implementation of real-time simulation and robot-team control interfaces (C2 user interfaces) when needed.

3.2 Initial technological infrastructure: AgentFly

In 2008, at the time of starting the first phase of Tactical AgentFly project, timewise the first project of the cluster, the *Agent Technology Center* had already an in-house developed technological infrastructure to start from. In the context of AgentFly project, described below, we developed an advanced technological platform for air traffic management and flight control of unmanned fixed-wing aircrafts. The initial design of the technology used for the Tactical AgentFly and Tactical AgentScout projects heavily relied on re-use and extension of proprietary, in-house developed technologies from the AgentFly project. In the following, we provide a brief overview of the individual software packages planned to be re-used, adapted, or integrated with the newly developed technological platform.

3.2.1 Application domain fundamentals: AgentFly overview

The AgentFly system developed in ATG between 2005–2011[1] is a technological platform facilitating development of mid and large scale simulations of autonomous aircrafts. As of 2008, the system was primarily aimed at investigation of issues in flight trajectory planning of various classes of fixed wing CTOL UAVs and their collision avoidance. I.e., a typical AgentFly scenario instance was configured with a landscape, a number of pre-configured no-flight zones and a number of aircrafts.

[1] As of writing this report, the project is still being actively developed, extended, and optimized.

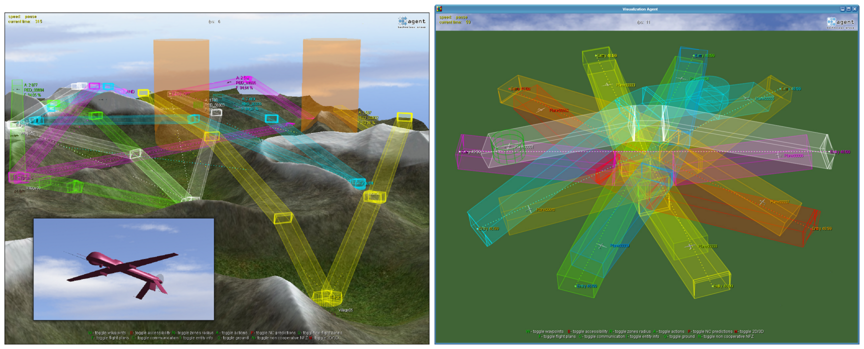
Fig. 3 Snapshot of an AgentFly scenario simulations.

Each aircraft was initialized with an initial position, space constraints on its flight trajectory envelope, speed vector (or landed on a ground runway) and a set of waypoints it should visit in particular times and fly over in particular speeds and heading vectors. The aircrafts were able to compute optimal flight trajectories spanning the waypoints, and at the same time respecting the constraints set up by the environment, together with the time, velocity and heading constraints. At the core of the system lies a decentralized algorithm running on board of the simulated aircrafts which was able to firstly detect conflicts between the trajectories of the planes and subsequently compute solutions to the conflicts in the whole multi-aircraft system. Figure 3 depicts a snapshot of an example scenario running in the AgentFly system.

Initially, the projects of the Tactical AgentFly and Tactical AgentScout cluster were supposed re-use the existing algorithms for trajectory planning and collision avoidance and extend them so that they could be integrated with algorithms for area surveillance and target tracking.

3.2.2 Multi-agent platform: Aglobe

The architecture of the AgentFly system relies on a state-of-the-art multi-agent platform Aglobe [29] and AglobeX Simulation its simulation extension (see Figure 4). Aglobe is an award-winning multi-agent platform, similar to *JADE* [4] or *Cougaar* [13], aiming at testing of experimental scenarios featuring agents' position and communication inaccessibility. The platform focuses precisely on modeling and development of decentralized mutli-agent systems. It provides facilities for residing agents, such as communication infrastructure, data store, directory services, weak migration function, deploy service, etc. Aglobe's implementation focuses on extremely efficient message transport layer and agent life-cycle management and thus features a very high level of scalability. From the perspective of a single agent, the platform provides two types of interfaces. Firstly, the MAS platform provides core functionalities including intra agent tasking, message communication, communication visibility, agent life-cycle, and migration. Secondly, it includes and handles

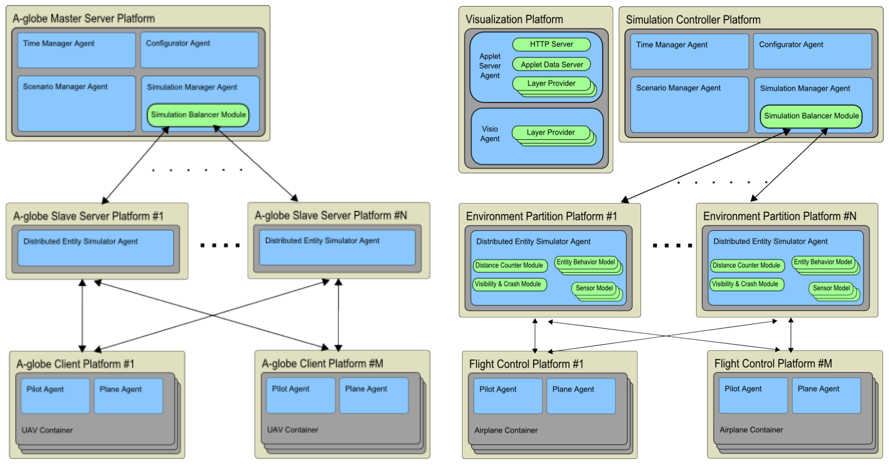

Fig. 4 Left, Aglobe multi-agent platform architecture. Right, AglobeX Simulation extension of the Aglobe platform. Both are depicted as they were instantiated in the AgentFly project as of beginning of the Tactical AgentFly project.

a number of services including directory/yellow pages, communication monitoring, and other. Due to its underlying architecture, the service interface is highly modular, as the individual services can be optionally turned on/off.

In Aglobe, agents reside in containers, which can be seen as logical groups of agents acting as a single entity from the perspective of the communication visibility subsystem. The containers run on platforms, where each platform is run within a single *Java Virtual Machine* (JVM), possibly on a single network node. Each agent is running asynchronously in its own computational thread and it is driven by an internal event queue. The events are strictly personalized for each single agent and are used both for self-tasking, as well as for processing external messages. The agents run and communicate in a fully asynchronous manner with respect to each other. The communication subsystem, the message transport layer, is optimized in various respects. All messages are pooled for more efficient creation and destruction of the underlying objects. User of the platform can configure whether the message payloads are sent by serialization or as references (available only within the same JVM). The message delivery mechanism also detects and derives efficient ways for how a message it to be delivered with regards to platform decentralization, communication capabilities of the underlyig network stack and target platforms platforms the agents reside on.

On the message transport layer, Aglobe is built a publish/subscribe mechanism featuring global topic messages. These are used for system communication, as well as widely employed by AglobeX Simulation simulator and other technology subsystems.

3.2.3 Environment simulator and scenario modeling: AglobeX Simulation

AglobeX Simulation (see Figure 4) is a simulator extending the Aglobe multi-agent platform. It aims at modeling and execution real-world 3D simulations including both static, as well as mobile units including towns, ports on one hand, as well as aerial and ground vehicles on the other.

The simulator is implemented as a number of coordinating Aglobe agents. The responsibilities of the agents are initialization, finalization and execution of the simulation. In the course of a simulation the agents compute next time-step evolution of the vehicle positions, as well as their simulate inputs to their sensors, detect collisions, prepare required information about the vehicles for the communication visibility subsystem and so on. At the time of using the platform in our projects, the core functionality of the AglobeX Simulation component was due to legacy reasons centralized and with respect to other subsystems functioned in a client-server mode. In its current incarnation, the simulator leverages the multi-agent paradigm as well as thus enables distribution of the whole simulator over a set of network nodes. The simulator's clients are agents or groups of agents united in a single MAS container controlling the simulated entities, which are representing their embodiments within the simulated environment. The agents communicate with the simulation server via topic messages. In one direction, the agents control their embodiments in the physical simulation and in the other direction, they are allowed to perceive the state or changes in the environment through their sensors.

The simulator consist of these particular agents:

Scenario manager agent: manages, preprocesses, and serves the configuration data to the simulation (see also Section 3.2.4), manages the simulation scenarios, their starting, finish, as well as monitors their execution.

Time manager agent: provides synchronized time ticks for the whole simulation.

Simulation manager agent: manages client agents and client containers, creates and removes simulated entities, mediates initial configuration to the entities (all this based on the configurations from Scenario manager agent).

Entity simulator agent: computes the step-wise evolution of the entity states (positions, orientations, etc.). Each entity is described by its behavior transforming the current state to a new one according to the commands received from the related client agent, resp. agent container. The behaviours together describe mechanics of the environment.

Distance agent: computes and stores distances among all entities in the environment.

Visibility collision agent: processes data from the Distance agent and i) provides them to communication visibility subsystem of Aglobe, and ii) uses them as a basis for entity collision detection.

Sensor agent: represents all monitoring and detection systems of the simulated entities (radars, cameras, etc.) and provides views to the simulation for the simulation agents based on time ticks from the Time manager agent.

All the server-side agents communicate among themselves via topic messages. The agents use topic messaging to in effect orchestrate and synchronize themselves as a single, centralized or distributed, simulation process.

3.2.4 Configuration, experiments, user interface and visualization

As already discussed above, in AglobeX Simulation, various types and courses of simulation are conceptualized into scenarios. Each scenario describes the initial state of the simulation instance, as well as the initial states of the controlling client agents. Each scenario also contains an initialization process for the related simulation. The scenarios are *Java* programs supported by mechanisms for XML-based configuration processing. A single scenario, together with its configuration initializes and executes a single simulation instance. Besides that, scenarios can also contain components gathering, processing, and storing results from experiments. A scenario can also describe a suite of particular experiments together with their automated configurations. The experimental results are collected from the simulated processes via topic messages or directly by monitoring the simulation evolution itself.

As a full-featured MAS platform, Aglobe provides various user interfaces (UIs) as well. The core one, the platform management UI. It contains a list of services and agents running on the platform and allows a user to run new, stop or examine services and agents running within the platform. To facilitate communication monitoring and debugging, there is also a communication sniffer tool. It enables tracking of inter-agent conversations and inspect the intercepted messages including their content. Optionally, agent implementations can come with their own application specific UIs. An example of such, is an informational UI provided by the Pilot agent of the AgentFly system.

AglobeX Simulation comes with its simulation-specific user interface. The Scenario manager agent provides dialogs for scenario selection and reset. The Time manager agent allows users to start, or stop simulation time, as well as set the simulation speed. Similarly, there are lists of the simulated entities, their respective agents, logging consoles, and others.

The last subsystem of the initial technological infrastructure is the Visio visualizer for the simulated world populated by agents. Visio is build on *Java3D* library [18] and *JOGL* library [21]. It enables 3D and 2D visualization of the simulated worlds, including the physical environment, simulated entities and additional information used during implementation, debugging phase and demonstrations. The visualized elements are organized in a hierarchical layout (used for the canvas drawing process) and a layered layout (used for turning on/off groups of visual elements). The visualization layers communicate with the rest of the system, i.e., client and simulation agents, by standard Aglobe topic messages.

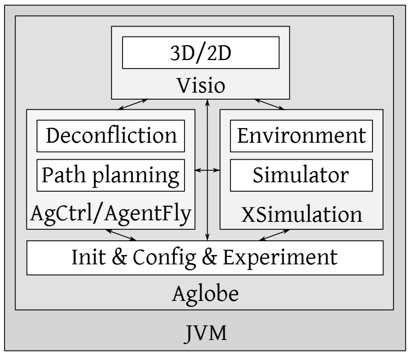

Fig. 5 Initial architecture of the technological infrastructure for the first phase of the Tactical AgentFly project based on AgentFly project (feature framing represent architectural composition, arrows represent coupling).

3.2.5 Agent behavior control: POSH

A significant requirement coming out of the initial analysis of the technological infrastructure underlying implementation of the Tactical AgentFly/Tactical AgentScout cluster was a need for a framework and an associated toolkit facilitating scripting of agent behaviors. The scenarios modeled in the project AgentFly did not require such complex behavior implementations. Early on, it became clear that for the projects Tactical AgentFly and Tactical AgentScout we need to bring in a new technology facilitating behavior scripting and integrate it with the rest of the system. Our initial idea was to leverage the state of the art frameworks for development of intelligent autonomous agents [5]. We chose to consider the *POSH* (Parallel-rooted, Ordered Slip-stack Hierarchical) reactive planning engine **[9, 8]**. *POSH* can be seen as a programming language allowing to execute programs encoding reactive action selection based on both the current state of the environment the agent perceives, together with its internal state, which can feature relatively complex data structures (beliefs). *POSH* follows the paradigm of behavior oriented design where behavior modules of a *POSH* agent provide the primitives for formulation of the plans. These are subsequently executed in run-time and represent the agent's deliberation about the particular action to be taken in the next step.

3.3 Initial system architecture

Figure 5 depicts the initial architecture of the underlying technological platform for the project cluster. It is based on the technologies from the AgentFly project. The platform stack runs within an instance of *Java Virtual Machine* (JVM) and the whole system was written in *Java*. Majority of the subsystems are implemented as agents running in the Aglobe MAS platform container. The Visio visualizer subsystemt is partially independent from Aglobe, since it incorporates a standalone part of the visualization engine.

All technological components of the system are configured with a simulation scenario descriptions and their respective XML configurations, initialization programs and experiment descriptions. These parts are depicted as *Init & Config & Experiment* subsystem. As already noted in the previous subsections, the AglobeX Simulation subsystem (abbreviated as *XSimulation*) runs as an orchestra of Aglobe agents as well. The simulation subsystem consist of the *Simulator* component managing time and simulation state. The *Environment* component manages simulated entities, sensors, and communication visibility.

The AgentFly system as a whole is represented by agents acting as aircraft pilots. A pilot agent implements a particular agent controls mechanism (abbreviated as *AgCtrl*). The component implements two main technologies. Firstly, it is a flight path planner and secondly, a decentralized collision avoidance mechanism, coined also deconfliction. These two components form the intelligent decentralized behavior of the individual aircrafts.

Finally, the visualization subsystem facilitates 3D and 2D visualizations, both based on Visio engine.

4 System implementation and experiences

The works on the projects Tactical AgentFly and Tactical AgentScout took part from early 2008 until mid 2011. Both projects were divided into two phases, each taking 12 months duration. Generally speaking, the initial phases of the projects were dedicated mostly to development and feasibility studies of the basic technological framework needed for realizing the main objectives. The successive iterations then focused primarily on the core objectives, namely research, development and empirical evaluation of various advanced multi-agent coordination algorithms.

In Subsection 3.1 we identified and summarized the main requirements on a technological infrastructure of the Tactical AgentFly and Tactical AgentScout project cluster. The remainder of this section discusses the details of their implementation as they were gradually incorporated into the code base. I.e., starting from the technologies underlying the AgentFly project, we discuss how the system architecture evolved by incorporating the individual requirements. During the process, we had to implement completely new modules and libraries and some of the modifications of the core elements of the infrastructure at the time resulted in significant shifts of its underlying philosophy. In result, some of the modifications were not backwards compatible with the original multi-agent platform anymore and led to significant architectural challenges. Alite is an agent simulation toolkit comprising all the general-purpose modules and libraries developed in the course of the works on the here described twin projects. Its gradual implementation could be seen as a consequence of tackling these challenges and, together with the gradual system evolution from a rigid single-philosophy embracing MAS platform to a pragmatic toolkit facilitating both a MAS platform composition, as well as rapid prototyping of the application specific functionalities. Recently, Alite grew out of providing only technological infrastructure for the Tactical AgentFly and Tactical AgentScout project cluster and it is already being used and extended in the context of various other projects ATG is involved in. While the initial design of the twin projects heavily relied on the Aglobe MAS platform and its AglobeX Simulation extension, the final phase of the Tactical AgentScout project was completely written using Alite toolkit and the libraries it provides. The remainder of this section tells the story of teh gradual shift of the underlying architectural philosophy on the background of series of extensions and requirements incorporation into a single system.

The treatment of the individual requirements follows the structure already introduced in the previous sections. I.e., starting with simulation and environment modeling issues, we continue by discussion of execution and configuration of experiments, through requirements and evolution of the agent behavior control mechanisms finally to discuss the implementation of user interfaces and visualization components of the system. Wherever possible, we respect the timeline of the work on the individual requirements.

4.1 Simulation and Environment Modeling

Due to the focus of the here described project cluster on tactical missions in urban environments, one of the first tasks the project team faced was extension of the AglobeX Simulation module with the ability to handle complex urban environments and landscapes. In particular, we extended it to handle 3D models describing buildings and the underlying street map providing the high-level environment abstraction the agents lived in.

With 3D buildings representing a town, agents moving along streets between these buildings and aircrafts flying over and screening the area by use of cameras, the environment modeling also needed to handle occlusions. To tackle this problem, we implemented an environment-specific simulation agent providing an occlusion sensor to the aircraft agents running in the system. As we discuss later, due to the inherent complexity of the AgentFly's AglobeX Simulation architecture, this design decision later turned out to be rather ineffective and after replacing the core simulation component by Alite specific module, the occlusions handling was implemented in a form of pure individual-agent level sensor.

The task of the aircrafts in the Tactical AgentFly project was to perform surveillance and tracking of urban areas with simulated human agents performing either relatively simple adversarial behavior (red force), or representing civilians in the town. The embodiments of ground entities were added into the environment as a new type of simulated entities (defined again by 3D position and direction) and integrated into the lists of simulated entities. They feature their own mobility models and behavior mechanics controlled by reactive planning engine(s) discussed later in this section.

The Tactical AgentScout project brought a new set of requirements which led to significant changes of the simulator. The incorporation of simulated UGVs, unmanned ground vehicles, resulted in a need to significantly adapt the model of the general physical dynamics of physical agents in the system and resulted in an incorporation of a 3D physics engine into the simulator. Execution of trajectory plans by autonomous ground vehicles in physical conditions can result in frequent plan invalidation. Besides dealing with terrain features, such as slopes, potholes, or stones on the ground, some of the corners are not traversable by the vehicle in its current speeds. In result, the autonomous cars have to move around the town by executing and continuous monitoring and adaptation of the path plans. As a part of the solution to the problem, we integrated JBullet an open source high-fidelity physics simulation engine [19].

To enable high-level control of the UGVs, which abstracts away the physical reality of the environment, we implemented discrete time movement of simulated ground vehicles on a street graph. In result, the high-level control algorithms steer the cars between waypoints on the street map (e.g., junctions). This development, together with the above described move from proprietary physical environment representation to a state-of-the-art technology for simulated physics led to abandonment of the AglobeX Simulation component from the system. This radical step was reinforced by the fact that AglobeX Simulation supports only simulated continuous space entity motion, however for many of our experiments only rough discrete motion on graph-based representations sufficed. In result, the move to pure Alite simulator implementation led to significant decrease of implementation, as well as debugging complexity of the individual experimental scenarios and allowed us to implement also simplified simulation model based on events instead of discrete time ticks (see also below).

To implement aircrafts performing close-up tracking of mobile targets, such as pedestrians and cars resulted in a need to incorporate reactively controlled aircrafts of various types into the system, be it conventional fixed-wing planes (CTOLs), or helicopters (VTOLs). Such UAVs are able to change their flight trajectory in a reaction to changes of movement patterns performed by the ground target. In the case of fixed-wing aircrafts, which cannot stop in mid-air, this problem results in a need to perform relatively complex flight patterns, such as various loops over the target. Together with a need to implement a fine-grained physical dynamic feedback control of helicopters respecting a realistic model of their physical movements, this led to a requirement to adapt the simulator to a much finer grained time resolutions.

Fixed-wing aircrafts feature a much simpler model of their physical dynamics and therefore can afford for longer delays between individual steering points. Due to relying on features such as this, the underlying AgentFly infrastructure turned out to be too rigid for our purposes. An implementation of the required fine time granularity control of physical entities would lead to a significant abuse of the platform and would have result in significant re-implementation of the mechanics integrating high-level planners and the low level physical control of the plane. In result, as already indicated above, we gradually departed from the AglobeX Simulation component and implemented the fine-grained simulator event loop in Alite from scratch.

The final implementation proved to be flexible and easily integrable with the other parts of the resulting system.

4.2 Evaluation of multi-agent coordination techniques

The main objective of the twin projects was to develop and most importantly experimentally evaluate various decentralized algorithms for coordination of multi-agent teams. This objective results in two basic requirements. Firstly, the simulator must be highly configurable in order to allow for high flexibility in terms of both, simulation experiment structure (number of agents, their various types, different initial conditions, etc.) and the executed scenario storyboard, i.e., the particular mission to be executed. Secondly, experiments comprise of large numbers of executed simulation instances and their runs have to reliably reproducible.

The Aglobe MAS platform, together with the AglobeX Simulation extension for AgentFly feature an XML-based configuration facility. This however turned out to be too rigid for the purposes of the Tactical AgentFly and Tactical AgentScout projects. The XML-based configuration files are interpreted by the simulator and the MAS platform so that the correct numbers of agents with correct initial conditions are instantiated in the system at the beginning of a run. However, such files allow for configuration of features foreseen during the system development and thus are inflexible with respect to the extremely broad range of scenarios implemented in the here described projects. As a part of departure from the Aglobe and AglobeX Simulation technology, we implemented a flexible configuration facility based on employment of dynamic programming language interpreters of Groovy [16] and Clojure [12]. Interpretation of full-fledged programs in run-time allows to configure any aspect of simulations in a concise and flexible way. In our context, the additional overhead of configuration script recompilation turned out to be relatively small and the corresponding minor slow-down of experiment runs is acceptable, as well.

An important aspect of simulation development is reproducibility of simulations. 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 limitations of the underlying hardware, such as CPU scheduling, or memory swapping on the limit of resource utilization, etc. To ensure reproducibility of experimental runs, we carefully considered and implemented the concept of *in vitro* simulation. That is, a simulation which controls all the aspects of the modeled system, or carefully accounts for those, which were abstracted away from. In particular, this means that the simulator has to have an ability to suspend and later resume the simulation process. Furthermore, it should have an ability to speed it up, or slow it down in response to e.g., resource utilization of the underlying hardware, so that race conditions and different results of process scheduling do not affect the simulation outcome. Finally, the random processes involved in the simulation must be also under the simulator's control so that the same sequences of random events are generated in two independent runs of the same simulation.

The vanilla MAS platform Aglobe is based on the assumption of full autonomy of agents which all run asynchronously in a truly decentralized fashion. This feature, while desirable in extremely large-scale simulations of air-traffic, turned out to be difficult to live with due to problems with the inherent non-determinism of the platform asynchronicity and the reliance of the simulation extension on discrete time ticking dynamics. In result, the need to run huge numbers of reproducible experiment runs turned out to hinge on the speed of simulation run execution and ability to make the runs deterministic on demand. To tackle this issue, we departed from the exclusive model of centralized discrete time ticks and implemented event-based simulation mechanism [2]. 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). However, at the same time the resulting Alite simulator still features the ability to run at real-time simulation speed for demonstration purposes. In the Tactical AgentFly and Tactical AgentScout projects, the newly implemented event-driven Alite simulator enabled complete synchronization of the simulated processes and thus facilitated high level of control over the simulated environment.

4.3 Scripting and Agent Control

As already mentioned above, realistic evaluation of various coordination mechanisms requires modeling of realistic behavior of the various actors comprising the environment the simulated multi-robot team acts in. As already discussed in Section 3, our initial idea was to employ a state-of-the-art agent programming framework *POSH*. Early on, however, this choice turned out to be difficult due to the fact that the programming system is a research prototype and was not ripe enough for straightforward integration into the project code base. Besides that, our initial feasibility study showed that the use of the framework by inexperienced users, programmers involved in our projects, was rather problematic also due to difficult *a priori* conceptual framework underlying the system.

The initial solution to our problem was implementation of Lightweight Reactive Planner (LRP) [27], a proprietary and relatively simple hierarchical reactive-control engine. While the component served us well in the first iterations of the project, later it also turned out to be inflexible due to its extreme simplicity. In result, we moved to implementation of *Belief-Desire-Intention* (BDI) agents implemented in an agent-programming framework *Jazzyk* [25]. *Jazzyk* features a high-level of modularity with respect to integration with 3rd party and legacy systems and allows their straightforward integration into the system in the form of agents' knowledge bases. This allowed us to use heterogeneous knowledge reasoning engines, such as *Prolog*, or object-oriented databases within a single agent system and at the same time maintain a high level of control over agent's behaviors, which were implemented as *Jazzyk* situated reactive plans.

4.4 User interface and visualization

The last, but at the same time one of the most important technological parts of the whole mosaic of subsystems are user interfaces and visualizers. Visualization components facilitate various debug, as well as demonstration views on what is going on within the simulation. Graphical 3D demonstrators constituted important deliverables of the Tactical AgentFly and Tactical AgentScout projects. To facilitate vivid and believable presentation of the research results we are using a composition of various 2D and 3D visual representations.

The requirements on simulation visualization can be divided into two types: i) global overview, and ii) informative details. At the same time, both types of visualizations had to be provided in 2D and 3D as appropriate with respect to the particular purpose. The approach to tackle this issue in the AgentFly project was based on *Java* 2D graphics and a protocol communicating with external 3D engine (namely *CrystalSpace* [14]). With regard to backward incompatibility of changes in newer versions of the 3D engine, this model was dropped. Later, still in the context of the project AgentFly, we implemented an in-house full-featured 3D and 2D visualization engine build on *Java3D* library [18] and later with several features based on Java native connections to *OpenGL* [21]. The engine uses concept of layers optionally showing various information from the running simulation. These layers are connected with data sources in the simulation by the Aglobe asynchronous messaging interface.

In the early phases of Tactical AgentFly and Tactical AgentScout projects, we re-used the working visualization engine from AgentFly. The engine was during the time extended only by additional 3D models and visualization layers for various trajectory and plan types (e.g, for UGVs and VTOLs).

However, with the already discussed departure from use of the AglobeX Simulation simulator, we were given an opportunity to improve upon the visualization technology and replaced the obsolescing visualization engine with *Java Monkey Engine* [20], a state-of-the-art *Java* 3D engine.

Additionally, we have created a simple 2D visualization engine primarily for debugging purposes of the new simulations and environments. The 2D engine was designed considering the common principles of Alite: flexibility, openness and modularity and became a universal tool for 2D visualizations of global overview of simulated environments. Unlike in the AgentFly project, we carefully crafted the system architecture so as to respect the principle of non-interference of visualization with the run of the simulation itself. This is mainly to preserve reproducibility of the *en masse* run experiments, which do not make use of the visualization code at all. The simplicity of the visualization engine also facilitated development of lightweight APIs, so that programmers could make use of it in a straightforward and flexible manner.

4.5 Alite

Alite *['eɪlaɪt]* [1] is a software toolkit aimed at simplifying implementation construction (not only) of multi-agent simulations and multi-agent systems, such as those implemented in the context of the Tactical AgentFly and Tactical AgentScout projects in general. The objectives of the toolkit are to provide highly modular, flexible, and open set of functionalities defined by clear and simple APIs towards easy rapid prototyping and fast implementation. The toolkit does not serve as a pre-designed framework or platform for a complex purpose, it rather associates number of highly refined functional elements, which can be variably combined and extended into a wide spectrum of possible systems.

The guiding principles underlying the Alite design are i) modularity, so that the system does not commit a developer to a particular definition of concepts such as *agent*, *environment*, etc., and ii) compositionality, 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 an association of highly refined functional elements providing clear and simple APIs, so that relatively complex multi-agent simulation scenarios can be put together rapidly.

Alite agents have access to simple interfaces to the environment (sensors and actuators), while their internal lifecycle is not bound by any *a priori* philosophy. Additionally, they can make use of various types of communication middleware interfaces allowing to model various types of intra-agent communication (synchronous, asynchronous, peer-to-peer, broadcasting, multi-casting, etc.). Additionally, Alite comes with libraries including various types of planners (reactive, deliberative) and multi-agent solvers (e.g., task allocators, solvers for problems, such as the distributed vehicle routing problem, etc.).

By its compositional nature, Alite provides means for both rapid prototyping, as well as high-level of elaboration tolerance of the implemented systems. E.g., once a simulation scenario, or a functional multi-agent system is put together from various components, application-level customizations and proprietary domain-specific mechanisms, it is very easy to replace one stock planner, or multi-agent solver by another one, as far as they share the underlying assumptions for their use.

Alite addresses the problem of MAS platform resilience in the face of a need to incorporate various *a priori* unknown future requirements by variability in composition of functional elements. The number of possible combinations include wide spectrum of system types, in contrast to a pre-designed frameworks as [29, 4, 13]. 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 more precisely fits the particular needs of the MAS application under development. The application can utilize one or more functional elements. As of writing this chapter, Alite provides:

common-event-queue: a general implementation of a temporal event queue and temporal events (can be used for event-based simulations, agent message queues, etc.).

common-entities: a general description of any entity in the system. An entity is defined only by its name (represent agents, simulated embodiements, etc.).

common-capability-register: a general implementation of a simple register of possible capabilities provided by entities (usable for directory services, register of simulation components, etc.).

communication: a component of communication interfaces and basic message transports (includes direct and asynchronous message transport, protocol abstraction, abstraction of communication modes, etc.)

initialization: a component defining basic interfaces for initialization scripts and configuration (includes a config-reader based on Groovy)

environment: a component of interfaces defining basic elements for simulated worlds (includes state storages, and bases for sensors and actuator interfaces).

simulation: a component mediating event-based simulation (it is based on the common-event-queue and enriches it by temporal control).

visualization: a set of component containing various visualizers or wrappers to 3dr party visualizing applications (includes 2D visualization, 3D visualization based on JME, wrapper to Google Earth, and others).

4.6 Architectural changes of the technological infrastructure over time

In the previous sections, we have discussed changes in particular parts of the infrastructure underlying the project cluster. Figure 6 depicts the overall system architecture as it chronologically evolved in the four iterations of the project cluster (two phases of the two projects).

The initial architecture for first Tactical AgentFly project come out of AgentFly with all technological features based on Aglobe (see Section 3.3).

In the next project, the Agent Control (*AgentCtrl* in the Figure) mechanisms were replaced by simplified implementation in Alite. The implementation was tested in newly added 2D visualizer, environment and simulation. All these components were initialized by Alite initialization mechanism (abbrev. *Init*) using optionally configurable parts and experiment description. The two systems were integrated by injection of positions from the new environment into the original environment and using time synchronization messages between the two simulators. The original system henceforth acted as a simulator for AgentFly airplanes and 3D visualizer, while the Alite simulator handled the ground mission simulation.

In the third project iteration all parts of the original system excluding the 2D and 3D visualizers were already abandoned. Simulations, environments and agent control, together with tested algorithms were run in the new system. Scripting mechanisms were also introduces to simplify implementation of particular agent control, mission scripting, and has enabled more general dynamic configuration and initialization. The integration of the two systems was based on propagating the envi-

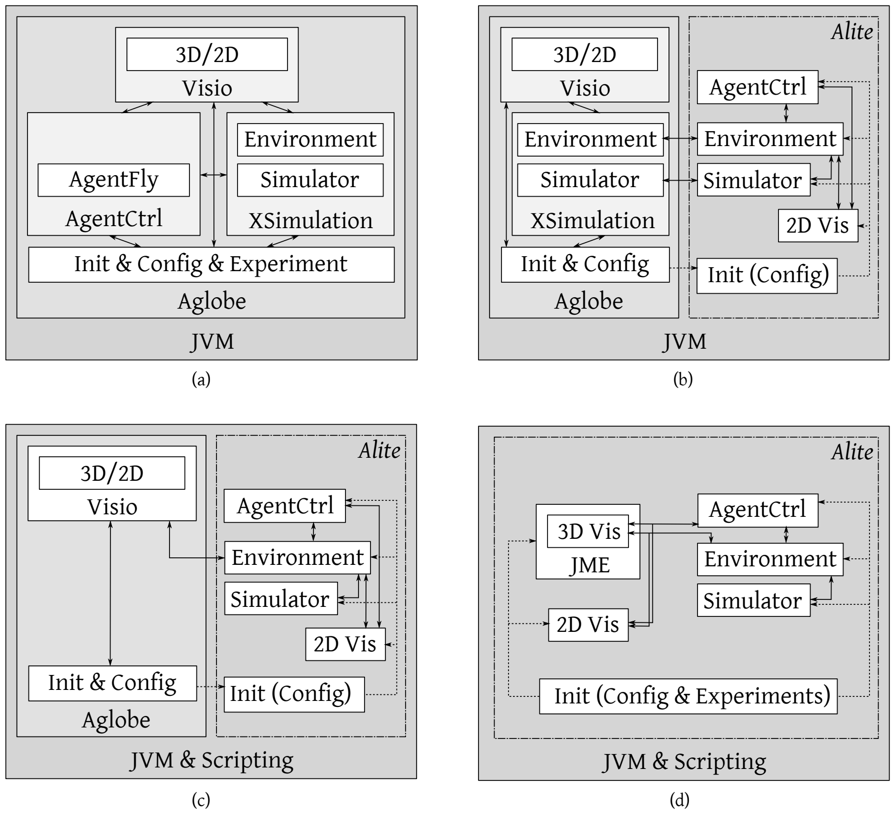

Fig. 6 Changes of the technological infrastructure over time. Firstly, the initial phase of the Tactical AgentFly project (a) was fully implemented using the Aglobe platform with AglobeX Simulation (abbrev. *XSimulation*). The initial infrastructure underlying the first phase of the Tactical AgentScout project (b) already partially abandoned some parts of the initial architecture. In the third iteration, namely the second phase of the Tactical AgentFly project, the original Aglobe a XSimulation are employed solely for visualization and configuration purposes. Finally, the second edition of the Tactical AgentScout project was fully implemented using the Alite toolkit. Feature framing represents architectural composition, solid arrows represent coupling, and dashed arrows initialization. Alite does not contain any platform code, thereby it only associates the composition of various utilized technological features (dash-dotted block).

ronment information (mainly positions of the simulated entities) into the original 2D/3D visualization through the Visio engine.

In the last project iteration, a new 3D visualizer based on *Java Monkey Engine* was added and allowed full shift from Aglobe-based system to more precisely fitting architecture covered by Alite Toolkit.

5 Critical analysis of the experience and lessons learned

To conclude the report of the previous sections on the implemented agent-based technological infrastructure underlying the Tactical AgentFly and Tactical AgentScout project cluster, let us identify and summarize the main lessons we learned in the process.

The here described project in fact comprises four more or less technologically separate sub-projects: the two thematically related projects Tactical AgentFly and Tactical AgentScout, both implemented in two phases of one year duration each. It can be said that to a large extent almost any software project can be implemented with almost any toolkit, development framework and programming language and following any kind of software engineering methodology. At the same time, however, some tools and methodologies fit some application domains better and lead to more efficient and more natural, resp. straightforward design and implementation process. Due to its structure and 4-year duration, we were given a unique opportunity to learn from past experiences, iteratively re-implement parts of the system and experiment with various configurations of its components and thus improve upon the previous experiences. In result, we not only identified the most important characteristics an ideal technological platform for research of coordination mechanisms in multi-robot missions in military operations should feature, but were also allowed to develop, implement, evaluate and even re-design various components of the technology as well.

From the technological perspective, our work can be seen as a four year long experiment with development of MAS platform, in particular including various simulators for mid and large-scale multi-robot systems comprised of heterogeneous robotic assets, such as CTOLs, VTOLs, UGVs and even simulated humans, different 2D and 3D visualizers, user interfaces and approaches to configuration and *en masse* experiment execution. We've learned that the two most important features around which the design decisions regarding the technological infrastructure revolve are the ability to support *rapid prototyping* and technologies enabling *efficient and fast empirical evaluation* of the implemented research algorithms. Generally, the discussion of architectural changes the platform underwent in the course of the project clearly shows that in time, we moved from an architecture embracing an instance of "one size fits all" philosophy towards open modular design comprising of a number of heterogeneous building blocks which can be composed into an application-specific design. In the following subsections, we discuss the individual aspects of the system and how the identified requirements on rapid prototyping and efficient empirical evaluation influenced their design.

5.1 Multi-agent platform

A software platform for implementation and run-time control of the individual agents running within the system is the most important component of the overall technological infrastructure. The project objectives dictated a need to compose various realistic simulation scenarios featuring different types of agents acting in various time paces, etc., so that we are able to test various research algorithms for multi-agent coordination. The sheer range of the scenario variety and at the same time the push towards rapid prototyping and fast research-implementation-evaluation turnover often resulted in clashes with the internal structure and the underlying philosophy of the currently employed multi-agent platform.

The lesson learned on this front is that often learned also in mainstream industrial software engineering. Namely, that in development of multi-purpose technologies, such as the one discussed here, it is often the case that an *a priori* application design philosophy, while being beneficial in the early stages of the project, tends to stand in the way of the development process in later stages. The stream of new requirements, not foreseen at the time of platform design, may sometimes diverge and even contradict some of the underlying principles of the design philosophy of the platform. An example of such was the inherent assumption of the Aglobe MAS platform with its AglobeX Simulation extension that the MAS application components should be modeled exclusively as agents, which communicate asynchronously. While this optics is well applicable in many applications, this design philosophy has vast consequences on the complexity of the application design, easiness of system debugging and reproducibility of experiment runs. Often implementing the simulator itself is more efficient using plain object-oriented programming principles with simple direct call method method invocation, rather than modeling even the system components as agents possibly running on different network nodes. Instead of working around this feature and thus abusing the underlying philosophy, in this case we rather decided to depart from this principle altogether and thus sped up the scenario development process.

In essence, the core lesson described above is that different MAS applications require different philosophies and technological features and the developers should be rather supported in compositional application development. In particular, this means that our resulting MAS technological platform based on the Alite toolkit keeps the MAS design open and comprising a number of complementary building blocks with clear interfaces keeps the door open to swift future redesigns of the application in question. Such crucial building blocks include various approaches and implementations of features, such as message passing asynchronicity, platform distribution among a number of network nodes, code mobility, agent lifecycle management, etc. Including a particular choice of these features in a single monolithic technological platform and its later extensions often tend to lead to software bloat and design decisions which constrain developers not because of some crucial issue, but due to respecting various interdependencies among the implemented features themselves. The lesson learned in the Tactical AgentFly and Tactical AgentScout cluster is that this situation should be avoided as much as possible. The shift the here described technological infrastructure underwent can be best described as

> *a move from a relatively rigid general-purpose MAS platform towards a toolkit facilitating rapid application-specific MAS platform construction.*

5.2 Environment simulation and scenario modeling

As already discussed in the previous subsection the range of simulation scenarios and modeled missions in our projects was rather large. Different scenarios aimed at evaluation of different multi-agent coordination algorithms often required very different environment features. While at times the environment could be modeled as a coarse grained graph structure with only interpolation of physical movements on the ground in the simulation, often we also needed high-fidelity environmental features including detailed physical landscape and building models. One of the lessons learned in the projects of the Tactical AgentFly and Tactical AgentScout cluster is that the stress on modularity and composability is crucial also with respect to an environment model, as well with respect to the particular model of time the simulator employs.

One of the most important parts of the simulation process is time handling. The underlying philosophy of the AglobeX Simulation simulator is a simulation model based on synchronous, constant-delay time ticks, which are asynchronously distributed to the simulated agents and other parts of the simulation. There are two common problems with such understanding of time counting in a simulation and both share their roots in the efficiency of the dependent simulation process. One is the temporal homogeneity and the other is causal homogeneity. While for many applications, the fine grained time model is directly employable, in our simulations it turned out to be rather inefficient. In particular, when a simulation contains agents working at various speeds, resp. being idle with different periods of time, the slowest time delay between two simulation ticks must correlate with the fastest agent in the system. This however leads to inefficiencies when an agent which normally exploits the fine grained time ticks becomes idle for a longer period of time. Essentially, nothing happens in the simulation, nevertheless the simulator is still forced to process all the homogeneous minuscule time ticks in between. Furthermore, synchronous time ticking simulations are inefficient for applications comprising large numbers of otherwise causally independent processes (e.g., flight of a UAV and activities of a ground soldier). In discrete time ticking simulations, such processes can become unnecessarily synchronized.

In the course of the projects development, we moved from synchronous time ticking to event-driven simulations. Yet, we were careful enough to maintain the ability of the simulator to switch to discrete time ticking whenever necessary. This allowed us to significantly speed up simulation time of scenarios which can be, without loss of generality, implemented in event-driven simulations. In result, we were able to shorten the research-implementation-evaluation cycle and ultimately also speed up the project completion. Additionally, in our experience, employment of event-driven simulations also simplifies the code required for implementation of agent deliberation and its interaction with the environment.

Finally, the requirement of reproducibility of simulation runs led us to attempts to realization of the concept of *in vitro* simulations. Since we were aware of this problem right from the beginning of the project it did not manifest itself in a significant manner in the course of our work. However, we consider it a noteworthy point to consider in simulator development. In simulations of multi-robot systems, the realization of *in vitro* simulators can become of an issue with growing demands on scalability of the system. While having all the aspects of the simulation under control of the simulator in a deterministic and synchronized manner is possible and manageable, the trade-off with growing scale of the system is its worse run-time performance, as well as possibly worse elaboration tolerance issues of the implemented system. The simulator can simply become too large component of the system featuring too many characteristics with underlying assumptions which significantly constrain simple and straightforward implementation of other parts of the system, e.g., agent behaviour control.

5.3 Experiments and configuration

One of the important lessons learned in our experience during the work on the twin projects is that in systems where the variation of future scenarios of its application cannot be easily foreseen, high level of configurability has to be implemented. Rather than relying on pre-defined configuration schemes, such as XML files, our decision to move to integration of full-fledged dynamic language interpreters turned out to be a good one. The cost of run-time configuration compilation at the beginning of the simulation run is negligible with respect to the overall simulation execution time. Furthermore, the gained benefit of practically limitless configurability of the simulation supports the rapid development principle and contributes to high level of modularity of the resulting simulations. These can be essentially constructed and initialized in the configuration scripts. In result, the tasks of simulation components implementation and scenario configuration become clearly separated what allows high flexibility in the development process.

The second important lesson to be learned from the work on the Tactical AgentFly and Tactical AgentScout project cluster is the stress on speeding up the research-implementation-evaluation turn-around. While the increased software platform modularity contributes to speeding up the first components of the cycle, attempts to speed up the runs of experimental setups improves upon the latter parts. It is also important to realize, that not only does faster experiments execution shorten the time needed for evaluation of the algorithms under investigation, it speeds up debugging and implementation part of the cycle as well.

In our particular project, the implementation of an event-driven simulation framework led to higher control over execution time of the simulation runs and speeding up the experiments evaluation. This experience however should be considered carefully in the context of our projects. For instance scenarios which would require extremely fine grained synchronous time would probably not benefit from implementation of event-driven simulation loop and also respecting of the requirements on implementation of *in vitro* simulations might become burdensome.

5.4 Agent behavior control

In terms of agent behavior control, again there are two issues to consider. On one hand it is the expressive power of the employed reasoning framework and on the other it is the modularity and integrability of its implementation. Furthermore, while some scenarios require quite a heavy-weight deliberation mechanism, in others, plain reactive control is sufficient. The lesson learned on this front leads to realization that if flexibility with respect to future applications is an issue, perhaps rather than employing a relatively heavy-weight agent deliberation framework (e.g., imposing BDI-style agent architecture), it might be more beneficial to use a simpler, but more general toolkit. I.e., it might be more flexible to invest an effort in learning and gaining experience with simple, but extensible deliberation models, such as e.g., finite state machines, rather than commit all future agent deliberation implementations to a particular intelligent agent architecture.

In our case, the choice of the *Jazzyk* language interpreter turned out to be a suitable choice for the more advanced simulation scenarios. The language is extremely simple, but at the same time it allows for compositional programming of agent behaviors. Additionally, it directly supports integration with the underlying simulator, as well as various knowledge representation technologies.

5.5 User interface and visualization

While user interfaces and visualization do not often stand in the focal point of prototyping in research projects, at this point, we would promote this issue and encourage rich visualization of simulations, especially in robotics research. Not only do realistic 3D scene visualizers provide attractive demonstrations, they also constitute a plausible basis for evaluation of the algorithms by non-experts. In our case, presentation of live 3D demonstrators and video sequences captured from various simulation runs proved to be beneficial in communication with both, the project sponsor, as well as 3rd parties.

The state of the art in open-source 3D scene modeling and animation technologies is at a stage, where straightforward use of the tools by non-expert programmers is easy. Having said that, it is of course important to design the interfaces to the 3D world visualization in a manner, which again stresses rapid prototyping of the resulting scenarios. In our case, integration of various types of 2D and 3D visualization engines in the Alite toolkit turned out to be relatively straightforward and cheap in terms of the involved implementation effort.

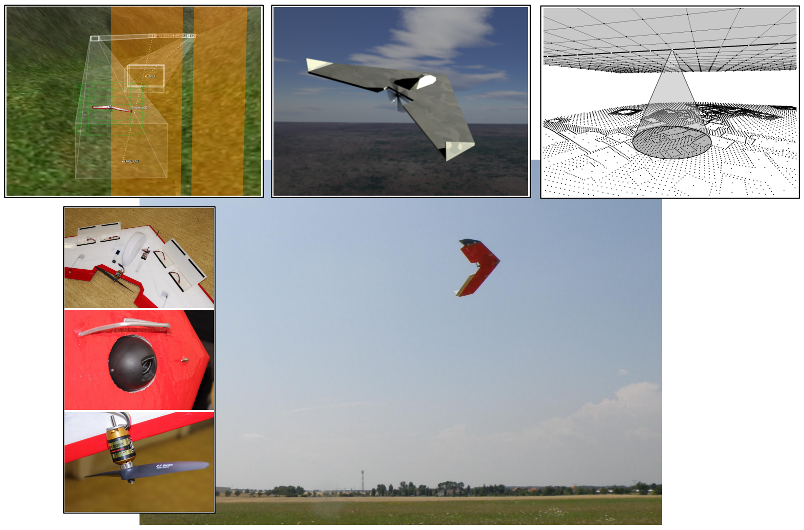

Fig. 7 Mixed-reality in the project AgentFly-In-Air, together with details of a *Procerus Unicorn UAV* test aircraft.

5.6 Towards AgentFly-In-Air

After completing the projects Tactical AgentFly and Tactical AgentScout, we collected a significant body of research results, experience and advanced its in-house technological platforms for development of mid and large scale simulations of multi-robot systems. A natural step along the above described research track was to move closer towards deployment of the developed algorithms to real hardware robots. The project AgentFly-In-Air (see Figure 7) aims exactly in this direction. The 18 months long project was started in mid 2011 and as of writing this chapter it is in active development to be completed by the end of 2012.

The foreseen demonstration scenario will involve a number of real, as well as simulated aircrafts performing a continuous surveillance of a pre-defined area and tracking of a number of mobile targets on the ground. Ideally, these will be embodied by real human subjects carrying GPS devices connected to a central simulation, resp. evaluation engine. By this experiment, firstly, we will demonstrate applicability and portability of the multi-agent coordination algorithms developed in the context of the above described projects to real hardware, and secondly, provide a proof-of-concept for the idea of *mixed-reality multi-agent simulation*. I.e., such, that besides a number of simulated agents, it will partly run in reality and will include real world robotic assets.

The imperative to port and deploy a selected subset of the algorithms developed in the context of its precursor projects in real hardware brought new requirements on the design of the overall system and raised new challenges as well. On the front of the application level functionality, the main issues are rooted in decentraliza-

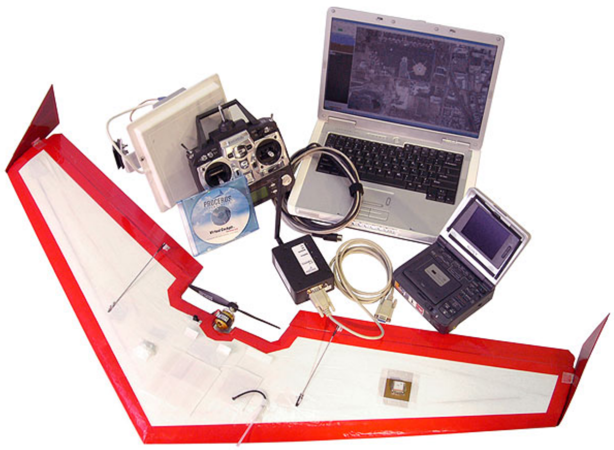

Fig. 8 *Procerus Unicorn UAV* test aircraft with accessories.

tion of the coordination algorithms and scalability of the algorithms to the on-board hardware and its resource limitations w.r.t. CPU power, battery usage, communication bandwidth, etc. Complementary to that, availability of only limited number of hardware robots, in our case two *Procerus Unicorn* aircrafts (see Figure 8), requires development of technology enabling development and execution of mixed simulations. That is experiments in which parts of the overall system are simulated, but significant parts are implemented in real hardware. In result, as a part of the project, we develop a technology allowing to model multi-agent systems in which all agents are equal w.r.t. their run-time characteristics and interfaces to the physical environment regardless of whether the agents themselves, resp. the environment are simulated, or deployed on real hardware. In result, the system should provide high level of modularity and facilitate gradual steps from fully simulated multi-robotic system through mixed simulation ultimately to pure hardware deployment.

The opportunity to employ the technological infrastructure developed in the context of Tactical AgentFly and Tactical AgentScout in a new setting embedded partly in real hardware, allows us to once again reconsider and critically analyze the use of the above discussed software components. One of the important issues arising from the need to both model, as well as to incorporate true robotic assets in the evaluation scenarios is the already discussed message delivery asynchronicity. While in the purely simulated scenarios, for the sake of respecting the *in vitro* simulation principles, we abandoned the real-world asynchronous inter-agent communication model, in the project AgentFly-In-Air we have to come back to it. In result, we consider to once again employ the Aglobe MAS platform as the underlying technology, but instead of using it to manage also agents' lifecycles, we will treat and use it purely as a decentralized communication middleware (see Figure 9). After all, a fine-grained management of agent lifecycle in terms of control of threads and processes running the logic of the individual agents becomes pointless in hard-

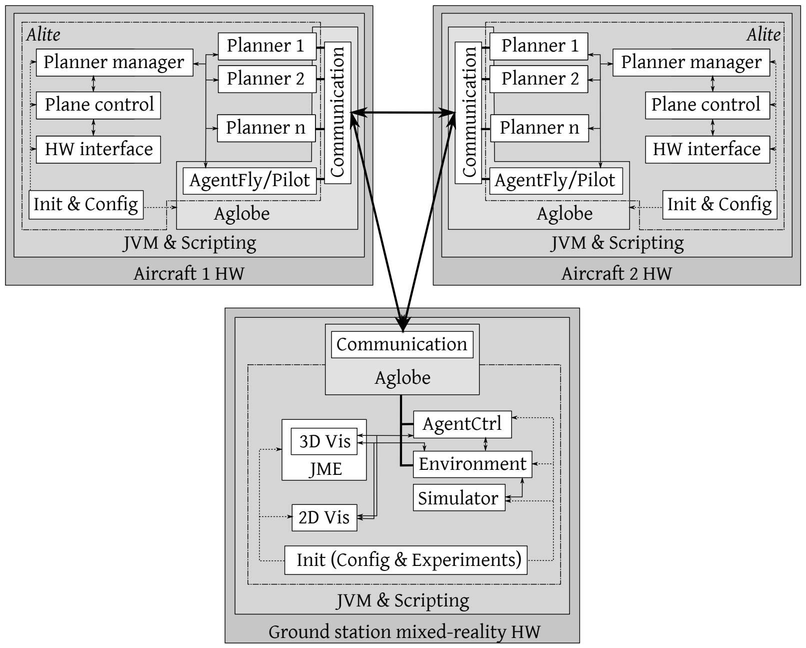

Fig. 9 Planned architecture for the project AgentFly-In-Air. The ground station runs a simulation based on Alite Toolkit. Communication with the real airplanes is mediated by Aglobe. The aircraft logic is a composition of new future Alite modules. As planners, modules from predecessor projects will be used and as a terminal planner Agentfly's Pilot Agent with path planning and collision avoidance will be integrated.

ware multi-robot systems. We are of course aware, that by this step we will loose the ability to execute evaluation in fully controlled environments akin to *in vitro* simulations, except with hardware robots.

In terms of environment modeling, we will face tasks to integrate real world sceneries into the simulator. It means, that to facilitate execution of the foreseen project demonstrator, we will have to be able to model a particular test ground within the simulation, including its landscape, terrain features, traffic network, buildings, etc. Implementation of new components facilitating various aspects of the mixed simulations, such as the landscape modeling, within the Alite toolkit will pose new implementation challenges in the project.

6 Future perspectives and final remarks

With the growing complexity of multi-agent applications and environments in which they are deployed, there is a need for development techniques that would allow for early testing and validation of application design and implementation. This is particularly true in cases where the developed multi-agent application is to be closely integrated with an existing, real-world system of multi-agent nature. Our work in the context of the here described project cluster including Tactical AgentFly, Tactical AgentScout and AgentFly-In-Air projects aims exactly at this objective.

Our experience in the course of the years 2008–2011 boils down to realization that due to the extremely wide range of scenarios for early testing and validation of algorithms, the scenario development and simulation technological infrastructure should remain extremely modular and feature a compositional architecture.

In the light of this lesson, we feel that rather than in development of special-purpose MAS platforms, the open challenges for the community lie in investigation of programming-framework-independent methodological guidelines for engineering of such multi-agent based software artifacts. Of course we understand that no unified MAS development methodology would fit the requirements of various application domains, however collecting and learning from experience with building such systems is still a realm in which not enough report are produced. Similarly to the MAS platform lesson highlighted in the critical analysis of the projects implementation, perhaps rather than aiming at a unified MAS development methodology (platform), our goal should rather be a set of rudimentary building blocks out of which such application-specific methodologies can be easily constructed on purpose.

To conclude the chapter, we would like to draw the attention to the notion of *mixed-reality simulation* and the methodological guidelines to be followed in development of such systems. Development and deployment of such complex multi-agent systems is a challenging task. Large numbers of spatially distributed active entities characterized by complex patterns of mutual interaction and feedback links give rise to dynamic, non-linear emergent behaviors which are very difficult to understand, capture and, most importantly, control. We argue that because of the complexity of the above-described types of applications, it is no longer possible to develop such systems in a linear, top-down fashion, starting from a set of requirements and proceeding to a fully developed solution. Instead, more evolutionary, iterative methodologies are needed to successfully approach the problem of development of complex multi-agent systems.

In [26], we make first steps towards tackling this open challenge and give a preliminary outline of the *simulation-aided design of multi-agent systems (SADMAS)* approach. SADMAS is a development methodology relying in its core on the exploitation of a series of gradually refined and accurate simulations for testing and evaluation of intermediary development versions of the engineered application. In particular, we propose and argue in favor of using a series of *mixed-mode simulations* in which the implemented application is evaluated against a partly simulated environment. Over time, the extent of the simulation will be decreasing until the application fully interacts with the target system itself. We argue that this approach could help accelerate the development of complex multi-agent applications, while at the same time keeps risks and costs associated with destruction or loss of the tested assets low. We believe that more research in this area is needed in order to better understand the core problems and issues stemming from deployment and evaluation of embodied multi-agent systems in real world scenarios, be it in industrial settings, or in military scenarios, such as those described earlier in this chapter.

Acknowledgements

The Agent Technology Center and the authors acknowledge the support of the here reported work reported by numerous projects awarded to ATG in the course of the years 2008–2011. The project Tactical AgentFly was supported by the grant W911NF-08-1-052 of U.S. Army CERDEC. The project Tactical AgentScout was supported under the grants BAA 8020902.A/W15P7T-05-R-P209 and W911NF-10-1-0112 by U.S. Army CERDEC. Additionally, the work was also partly supported by the grant MSM6840770038 of the Ministry of Education, Youth and Sports of the Czech Republic.

References

1. Alite project website. `http://agents.cz/projects#alite`.
2. Jerry Banks, John Carson, Barry L. Nelson, and David Nicol. *Discrete-Event System Simulation (4th Edition).* Prentice Hall, December 2004.
3. Roxana A. Belecheanu, Steve Munroe, Michael Luck, Terry Payne, Tim Miller, Peter McBurney, and Michal Pěchouček. Commercial applications of agents: lessons, experiences and challenges. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, AAMAS '06, New York, NY, USA, 2006. ACM.
4. F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. JADE a white paper.
5. Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors. *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer, 2005.
6. Christoph Borst, Thomas Wimbock, Florian Schmidt, Matthias Fuchs, Bernhard Brunner, Franziska Zacharias, Paolo Robuffo Giordano, Rainer Konietschke, Wolfgang Sepp, Stefan Fuchs, Christian Rink, Alin Albu-Schaffer, and Gerd Hirzinger. Rollin' justin - mobile platform with variable base. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 1597 –1598, may 2009.
7. B. Bošanský, V. Lisý, M. Jakob, and M. Pěchouček. Computing time-dependent policies for patrolling games with mobile targets. In *Tenth International Conference on Autonomous Agents and Multiagent Systems (to appear)*, 2011.
8. Cyril Brom, Jakub Gemrot, Michal Bída, Ondrej Burkert, Sam J. Partington, and Joanna J. Bryson. Posh tools for game agent development by students and non-programmers. In *University of Wolverhampton*, pages 126–133, 2006.
9. Joanna J. Bryson. The behavior-oriented design of modular agent intelligence. In *Agent Technologies, Infrastructures, Tools, and Applications for e-Services*, pages 61–76. Springer, 2003.
10. Lukáš Chrpa and Antonín Komenda. Smoothed hex-grid trajectory planning using helicopter dynamics. In *Proceedings of International Conference on Agents and Artificial Intelligence (ICAART)*, volume 1, pages 629–632. SciTePress, 2011.
11. Lukáš Chrpa and Peter Novák. Dynamic trajectory replanning for unmanned aircrafts supporting tactical missions in urban environments. In Vladimír Marík, Pavel Vrba, and Paulo

Leitão, editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 6867 of *Lecture Notes in Computer Science*, pages 256–265. Springer Berlin / Heidelberg, 2011.
12. Clojure project website. `http://clojure.org/`.
13. Cougaar project website. `http://www.cougaar.org/`.
14. CrystalSpace project website. `http://www.crystalspace3d.org/`.
15. Fukushima robot operator writes tell-all blog (news article). `http://spectrum.ieee.org/automaton/robotics/industrial-robots/fukushima-robot-operator-diaries`.
16. Groovy project website. `http://groovy.codehaus.org/`.
17. High-tech goes into action in disaster zone (news article). `http://www.msnbc.msn.com/id/9240563/ns/technology_and_science-science/t/high-tech-goes-action-disaster-zone/`.
18. Java3D project website. `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html/`.
19. JBullet project website. `http://jbullet.advel.cz/`.
20. jMonkeyEngine project website. `http://jmonkeyengine.com/`.
21. JOGL project website. `http://jogl.dev.java.net/`.
22. Antonín Komenda and Peter Novák. Multi-agent plan repairing. In *In Proceedings of Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities, DMPOUW 2011, IJCAI 2011 colocated workshop*, 2011.
23. Mining sector embraces sci-fi future (news article). `http://finance.ninemsn.com.au/newsbusiness/aap/8259880/mining-sector-embraces-sci-fi-future`.
24. Michael Montemerlo, Sebastian Thrun, Hendrik Dahlkamp, David Stavens, and Sven Strohband. Winning the darpa grand challenge with an ai robot. *Artificial Intelligence*, 21:982, 2006.
25. Peter Novák. *Jazzyk: A Programming Language for Hybrid Agents with Heterogeneous Knowledge Representations*, pages 72–87. Springer-Verlag, Berlin, Heidelberg, 2009.
26. Michal Pěchouček, Michal Jakob, and Peter Novák. Towards simulation-aided design of multi-agent systems. In *Post-proceedings of the eighth international workshop on programming multi-agent systems, ProMAS 2010, LNAI, Vol. 6599*. Springer-Verlag, 2010. (in print).
27. Michal Pěchouček, Michal Jakob, Eduard Semsch, Dušan Pavlíček, and Vojtěch Eliáš. Intelligent software agent control of combined UAV operations for tactical missions - final report. Technical report, Agent Technology Center, Department of Cybernetics, FEE Czech Technical University in Prague, 2009.
28. Eduard Semsch, Michal Jakob, Dušan Pavlíček, Michal Pěchouček, and David Šišlák. Autonomous uav surveillance in complex urban environments. In Conor McGann David E. Smith Maxim Likhachev, Bhaskara Marthi, editor, *Proceedings of ICAPS 2009 Workshop on Bridging the Gap Between Task and Motion Planning*, pages 63–70, Greece, September 2009.
29. David Šišlák, Milan Rollo, and Michal Pěchouček. A-Globe: Agent platform with inaccessibility and mobility support. In Matthias Klusch, Sascha Ossowski, Vipul Kashyap, and Rainer Unland, editors, *Cooperative Information Agents VIII*, volume 3191 of *Lecture Notes in Computer Science*, pages 199–214. Springer, 2004.
30. US military's UAV missions increasing (news article). `http://www.armedforces-int.com/news/us-militarys-uav-missions-increasing.html`.
31. Jiří Vokřínek, Peter Novák, and Antonín Komenda. Ground tactical mission support by multi-agent control of uav operations. In Vladimír Marík, Pavel Vrba, and Paulo Leitão, editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 6867 of *Lecture Notes in Computer Science*, pages 225–234. Springer Berlin / Heidelberg, 2011.
32. Willow Garage website. `http://www.willowgarage.com/`.
33. Michael Winikoff. Implementing commitment-based interactions. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, pages 128:1–128:8, New York, NY, USA, 2007. ACM.
34. Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.