Validation of Autonomous Concepts using the ATHENA Environment

Christophe Guettier(1), Bruno Patin(2), Jean-François Tilman(3)

(1)Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto CA 94022
Email: guettier@xerox.parc.com

(2)Dassault Aviation,
78, quai Marcel Dassault
92552 Saint-Cloud, France
Email: bruno.patin@dassault-aviation.fr

(3)Axlog Ingénierie
19-21, rue du 8 mai 1945
94110 Arcueil, France
Email: jean-francois.tilman@axlog.fr

1 INTRODUCTION

ATHENA is a simulation framework for supporting a prototyping design process of autonomous systems, such as unmanned aerial vehicles (UAV), unmanned undersea vehicles (UUV) or Autonomous Spacecraft Constellations (ASC). The use of several autonomous systems in the same command and control loops has become a challenge for both aeronautic and space industries through UAV (Fig. 1) and ASC. First, those systems must fulfil classical system engineering requirements such as timeliness, reliability, safety or survivability. Second, they must achieve difficult operations that are traditionally relying on human intelligence. The need for an in-situation tool supporting autonomous strategies verification and validation is fundamental for the design of these systems. The simulation framework ATHENA has been specifically designed for rapid prototyping and user-friendly assessment of automatic reasoning methods and multi-agent architectures relevant to the autonomy problems. Such a framework must be generic to a great number of domains and its own design modular enough in order to prototype both the assemblage of autonomous systems and their environments.

The state of the art in multi-agent systems provides interesting concepts for combining leading-edge automatic reasoning (based on constraint programming, Boolean solving) and distributed systems. This field of investigation has been the subject of many promising experiences for management autonomous unmanned vehicles.

To behave with limited human supervision, an autonomous agent must construct abstract representations of its own environment and situation. Based on this knowledge, the system can reason on its own behaviour in order to react efficiently and safely. Furthermore, the increasing complexity of those on-board systems gained an order of magnitude by considering complex coordination and collaboration schema for managing autonomously a set of vehicles. Then, an agent must also consider other agents situation and behaviour, leading to coordination and collaboration protocols involving knowledge maintenance and revision. Lastly, distributed algorithms must support the execution of those high-level decisional protocols.

This work was initiated and firstly funded by Dassault Aviation. A club was created in order to support the generic development. Three companies are involved in it, Axlog Ingénierie, Prolexia and Dassault Aviation. The goal of this club is to share the cost of this effort and give the tool the opportunity to be used in other domains than the one intended at its beginnings.

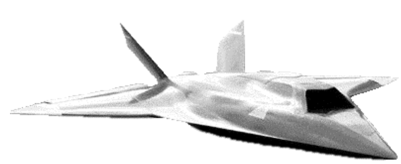

Fig. 1. The Dassault Aviation "Petit Duc" UAV

2 AN OVERVIEW OF ATHENA

The main areas to be addressed by a designer in the prototyping, the modelling and the assessment of autonomous system are the on-board system capabilities, the agent environment within this on-board system and the agent abilities within its environment. The solution proposed in ATHENA is to separate the representation of all domain specific problems in three layers: the physical layer takes into account the representation of our physical world, the on-board system layer emulates processes execution by on-board computing devices, and the agent layer, integrated in these processing systems, takes into account specialised processes that model agent. Each domain relies on generic meta-models that can be specialised by the end-user according to specific physical interactions, on-board system design choices and agent-based architectures.

2.1 Environment Layer

This layer defines the physical part of the simulated world, using continuous and discrete simulation models extracted from scientific theories (electromagnetic laws, fluid dynamics, flight dynamics, gravitation, etc). In order to elaborate a simulated state of the physical world at a given instant (Fig. 2), models of physical laws are applied to the previous states among the world history. Continuous laws can be abstracted by state-based finite automata in order to simplify the simulation or to model non-continuous phenomena and physical objects interactions. The discrete time representation underlying the simulation involves necessary approximations that can be managed in several ways by the user, especially at the limits of physical models. For example, models interpolation or extrapolation functions can be provided for discrete and continuous models. Also, the user can adapt a sample period, characteristic of its physical model.

2.2 On-board Processing System Layer

In order to model the on-board processing system that deals with computing, memories and communication capabilities, we introduce process components. This provides computing capability when integrated to a physical object representation. A process is composed with real-time tasks encapsulating on-board hardware or software functionalities. The default execution model statically schedules tasks, however more complex models (including dynamic policies) can be provided by the designer, such as FIFO ordering, Highest Priority First heuristics (Deadline Monotonic, Rate Monotonic), Earliest Deadline First. Distributed execution models can also be easily integrated (real-time transactions, serializable distributed execution). In particular, this facilitates the prototyping of Integrated Modular Avionic.

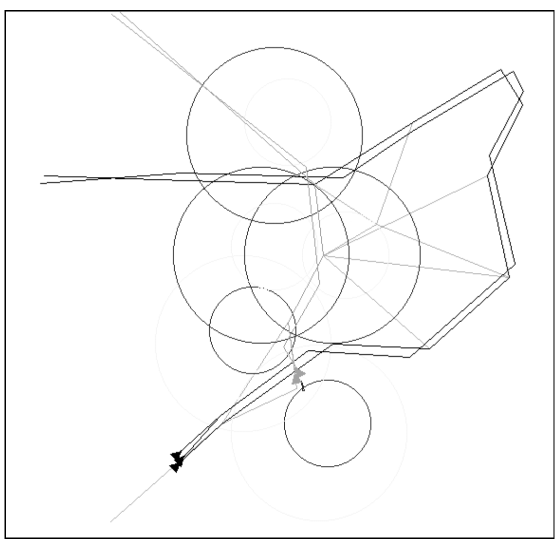

Fig. 2 Instant view of a tactical environment
Two flying formations perform a mission over the enemy territory, defended by surface-air systems.

2.3 Agent Behaviour Layer

The definition of decision and reasoning functions relies on a Prolog interface with the on-board processing system layer. This Logic Programming tool allows the designer to define knowledge-based algorithms, using the Solving Logic Demonstrator. By interpreting constraint predicates using mathematical algebra, the extensions of the language to Constraint Logic Programming can model efficiently combinatorial and discrete optimisation problems. In addition to the goal solving, basic agent functions have been developed such as knowledge revision and knowledge maintenance, using a dynamical management of Prolog facts.

Other tools can be used to implement Agent behaviour such as PROCOSA [1], an ONERA implementation of Petri net, or specific agent language (JACK for example, see [2]).

2.4 Specifying the Simulation

The simulation of physical objects, on-board processing systems and reasoning agents can be specified using an Architecture Description Language (ADL). This language is intended for the end-users that are not necessarily software experts. Thus, this ADL is simpler than other ones, generally oriented toward software engineers.

A simulated object is a composition of parameters, which contain data (either discrete and continuous types), states and transitions to represent an automaton, interactions between the parameters, and processes that represent the on-board processing system. The ADL gives a representation of all these elements. To allow creation and reusability of large and complex parts of simulation, the ADL provides prototypes. The concept of a prototype is near the concept of class in oriented object languages. A prototype can be instantiated and can be used as an ancestor to define another prototype. Fig. 3 gives an example of description of a prototype. Fig. 4 shows the instantiation of this prototype

By parsing the description files, ATHENA composes automatically the simulation and distributes its execution over a Network of Workstation (NoW). ATHENA also provides a specification language for the easy composition and integration of heterogeneous data-flow graphs, based on functions relevant to the autonomy domain (such as sensing, data-fusion, reasoning activity and actuation control). Each function can use a specific programming paradigm among signal processing filters, logic and constraint programming or other mathematical tools. This specification will facilitate further high performance optimisations by using the state of the art in parallel computation and optimising compilers.

Fig. 3. Description of the simulation

The simulation contains three instances of the previously described aircraft

```
INSTANCE Aircraft leader;
INSTANCE Aircraft wingman1;
INSTANCE Aircraft wingman2;
```

Fig. 4. Description of prototypes with ADL

The Aircraft prototype contains an automaton to detect crashes when the altitude becomes null. The UAV prototype inherits from Aircraft and contains an interaction to control its altitude

```
PROTOTYPE Aircraft
  PARAMETER double altitude = 10;
  TRIGGER crash: altitude dequals 0;
  STATESET state {flying, crashed}=flying;
  EVENT accident {crash = 1};
  TRANSITION flying : accident -> crashed;
END;

PROTOTYPE UAV IS Aircraft
  INTERACTION altitudeControl:
    IF flying,
    changeAltitude(altitude);
END;
```

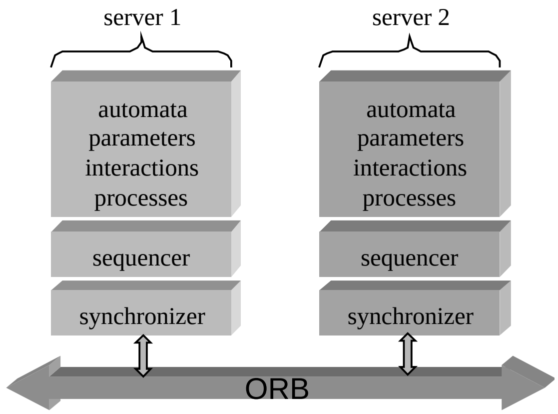

Fig. 5. Global view of the simulation engine

3 SIMULATION ENGINE

The simulation uses one or several simulation processes distributed over a NoW. As shown by Fig. 5, such a process is composed with a distributed synchroniser, a local sequencer, and a container for different kind of simulation elements (automata, parameters, interactions, processes, etc.). Each local sequencer schedules the activity of contained elements (methods activation's, events propagation, …).

Distributing the simulation over several processes is useful for several reasons. First, many users can interact simultaneously with various physical obj ects. Second, optimising the workload over several workstation can improve simulation response time, even at coarse grain. Lastly, dedicated architectures can be useful for executing specific functions (such as 3D visualisation, specific I/O devices, specific computation software…).

3.1 Distributed Clock Synchronisation

The main concept is to maintain a logical time for each physical, on -board system or agent component, in order to achieve a consistent simulation. A distributed synchronisation method, based on the asynchronous Welch algorithm [3] maintains a common logical date between all the simulation servers.

The causal dependencies between read and write operations on parameters are guarantied by a non zero delay between a write operation and the following read operation on the same value: when the logical time is t, all reads must be done on values written at t -1 at the latest, and no writing can be done on values in the past. All the communications are supported by an Object Request Broker (ORB) which al lows the engine to access parameters without worrying about their localisation.

3.2 Modularity

The end-user can introduce specific data types and functions into its simulation. Indeed, Athena provides a mechanism to plug new components at execution time. This facility aims at giving more flexibility to tackle problems specific to each industrial domains. For example the user can construct particular data types to handle a flight plan in an aeronautic simulation, or the spherical coordinates implemented with quaternions in a satellite simulation.

3.3 Visualisation Facilities

In order to define a simulation, we have to work on files that, even with a simplified language, become difficult to manage by their sizes and their numbers (a typical simulation can include a round one hundred prototype or more). To go through this preparation an interface has been developed that helps manage these files. Each file is linked to an icon and when you introduce a component by this icon, you include a prototype file in the simulation definition.

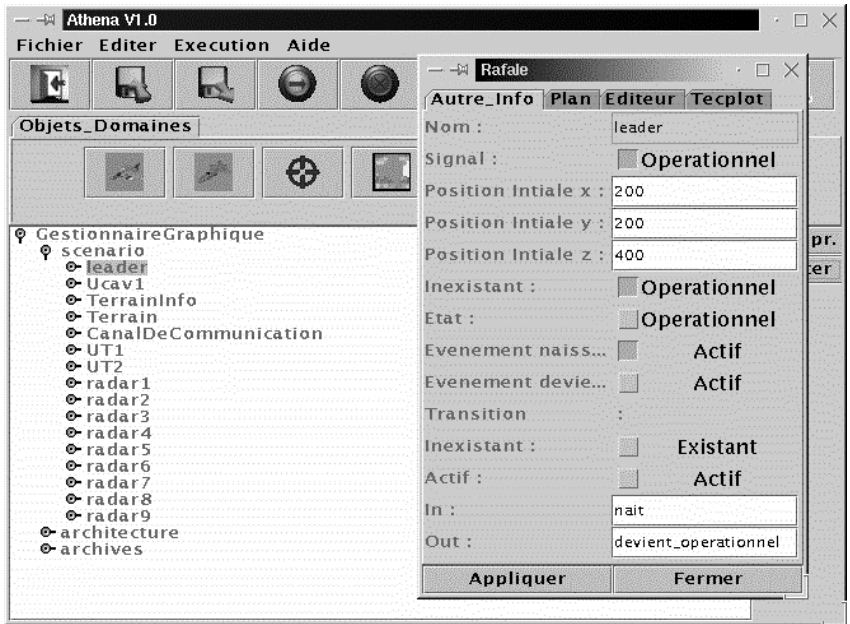

Fig. 6 Configuration tool

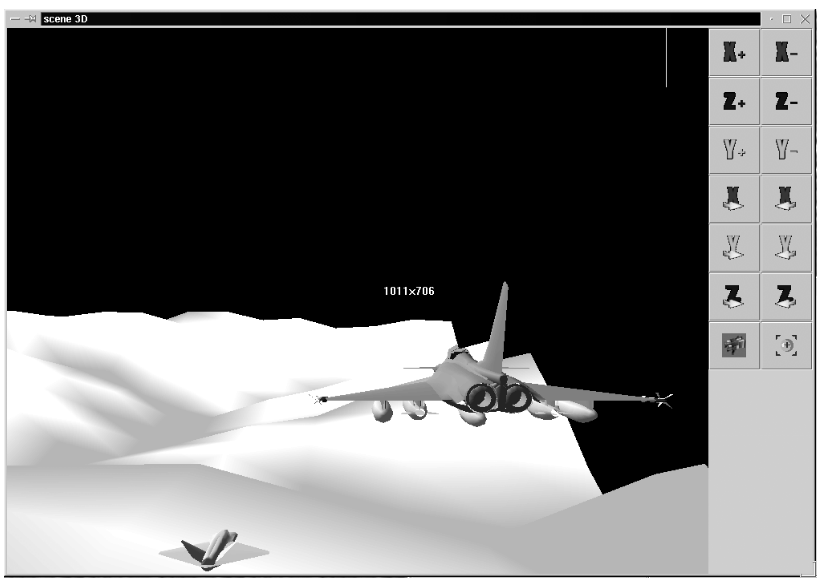

Fig. 7. 3D visualisation example

In order to interact with the simulation, a generic 3D interactive interface has been developed, supported by a JAVA based 3D visualisation engine. It is possible to associate to any object a visual 3D representation as well as defining user updates on its own state. A generic method has been developed in order to map easily a graphical behaviour onto any simulated object. This method does not require any modification inside the visualisation engine. The designer must only provide a graphical module, loaded dynamically by the visualisation engine.

The post-processing phase is covered by specific tools. Each ones use the recording of th e data that has been also made during the preparation. As an example, we use TECPLOT© on the position record.

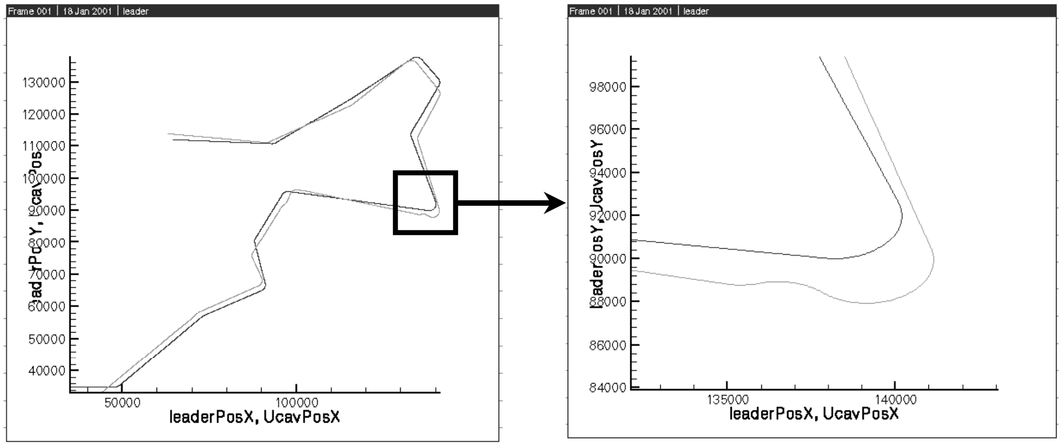

Fig. 8. 2D visualisation example

4 ON-BOARD AUTONOMY EXAMPLES

This simulation-based approach is illustrated by two examples , relevant to the aeronautic and space domains. Instead of adapting different libraries of simulation components, the framework facilitates their generalisation, and leads to their progressive share over several applications.

4.1 Aeronautic Mission

A preliminary study of an air-surface raid is the framework for the aeronautic demonstration (Fig 2). We consider a first flying formation of one leader plane and two strike UAVs used as wingmen and following the leader. The formation must fly to a given target in t he enemy territory and come back. The enemy territory is defended by ground surface -air systems composed with surveillance radars, tracking radars and missiles launchers. A second flying formation, composed by two planes, follows another flight plan over the enemy territory

When the opponent activates a ground defence that was unknown at the mission preparation stage or when the coordinate of a ground defence was not sufficiently known, the existing plan is no longer feasible. At this point, with a minimum of communications (we want to keep the strike patrol undetected), a new plan must be delivered to take this new threat into account. The leader produces this plan. It reflects a new collaboration strategy between UAVs including the management of the attack. Each wingman receives this plan and deduces its own new one (fig. 9).

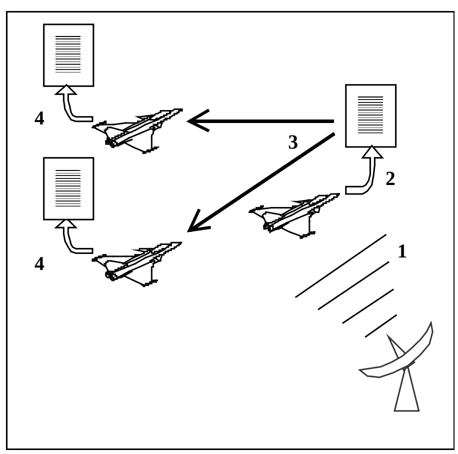

Fig. 9. Replanning in the aeronautic mission

The threat is detected by the formation (1), the leader computes a new flight plan (2) and transmits it to the wingmen (3). Each wingman computes a local flight plan from the given plan (4).

Generic components simulated include the electro-magnetic environment and the geometric environment. Domain components simulated include sensors (electro-magnetic, position, …), actuators (flight, fire, communication, …) and controllers. The automatic planning methods are included thanks to the on board processing system layer discussed above.

4.2 Mission for a Gamma Ray Burst Observations

This experiment simulates the behaviour of a distributed observation system orbiting around earth, based on multiple spacecraft. The goal of the mission is to detect and observe gamma ray bursts in various bandwidths. The ASC is maintained in a steady position in spite of regular scanning manoeuvres. Each spacecraft is assimilated to an agent that scan different parts of the sky, according to a long-term plan (Fig. 10). When a burst occurs, short-term and mid-term collaboration schema must be immediately performed (Fig. 11) in order to maximise the benefit of distributed sensors. Both spacecraft manoeuvres and payload schedules have to be coordinated to execute multiple bands observation sequences in the right direction.

To fulfil the required reactivity, the multi-agent architecture maintains possible collaboration and coordination plans while scanning, before the burst. The time horizon are very heterogeneous, some spacecraft must be pointed in few seconds to observe the gamma ray burst, while X-ray observations can be performed in a day and a week for the visible bandwidth. Therefore, the part of the plan to be maintained while scanning must be accurate for the few first seconds after the blast and eventually completed on-line for the remaining hours and days. Our approach combines constraint model based solving and multi-agent architectures [4][5]. The architecture relies on a constraint solver that consider altogether multiple models for solving the ASC planning. Generally NP-complete when tackled separately, models can be summarised as follow:

- Payload to bandwidth allocation (payload configuration, bandwidth to observe: Gamma, X, visible with associated deadlines);
- Manoeuvre duration (function of the difference between the current and target orientations);
- Energy consumption, such that spacecraft that have more energy would be able to support more active control.

A plan is solved for each possible section of the sky where a burst can occur. So doing, the planning workload can be parallelized over spacecraft, each of them being in charge of a sections set. An incremental search technique starts to give solutions from the shorter horizon (few seconds) to longer ones. Therefore, the burst can occur when the plan is not completed.

Based on spacecraft reactivity, propulsion and instrument resources as well as manoeuvres timeline, plans are immediately executed when the burst occurs. Those plans can maximise the ASC reactivity or the amount of observations performed in a time period. Components simulated include on-board processing, distributed coordination and automatic planning methods.

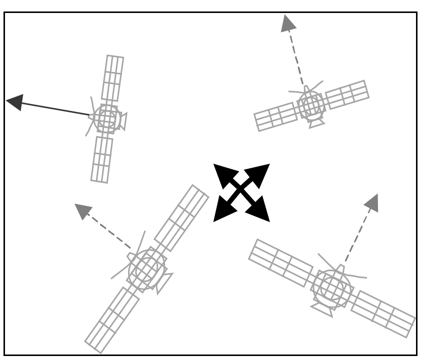

Fig. 10. Coordinated planning, each spacecraft sense a part of the sky and provide possible future plans

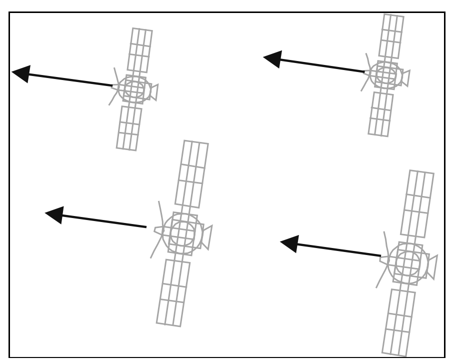

Fig. 11. Immediate Plan Execution, no planning required for the first seconds of the burst observations

5 CONCLUSION

We conclude by giving some experimentation synthesis and a demonstration will be available during the Workshop. Further works will focus on extending the experimentation field to larger and heterogeneous multi-agent architectures, and will address UUV. Parallel computing techniques will also be introduced in order to achieve high performance onto Cluster of Workstation architectures or parallel computers. To increase the modelling power, the ADL will be extended using other specification languages (such as Esterel [6]) to prototype the applicative behaviour of Integrated Modular Architecture more easily. Furthermore, high level agent built-in predicates and interfaces will be investigated to provide generic collaboration (introduction of a collaboration language such as KQML) and coordination algorithms. Lastly, interfaces with the High Level Architecture (HLA) [7] and other constraint solvers will also be implemented. At the end, Athena allows prototyping of all the aspects of a whole mission, with autonomous systems, supervision by an external user.

6 REFERENCES

[1] http://www.cert.fr/fr/dcsd/CD/CDPUB/PROCOSA/

[2] N.Howden, R. Rönnquist, A. Hodgson and A. Lucas, “Jack intelligent agents: Summary of an agent infrastructure”, Fifth international conference on autonomous agents, Montreal, 2001.

[3] Nancy A. Lynch, *Distributed algorithms*, Morgan Kaufmann Publishers, Inc., 1997.

[4] Eric Bornschlegl, Christophe Guettier and Jean-Clair Poncet, “Automatic Planning for Autonomous Spacecraft Formations” in Proc. of the NASA International Workshop on Planning and Scheduling for Space, San-Francisco, 2000.

[5] Christophe Guettier and Jean-Clair Poncet, “Multi-levels Planning for Spacecraft Autonomy” in Proc. of the International Symposium on Artificial Intelligence, Robotic and Automation for Space, Montreal, Canada, 2001.

[6] Gérard Berry, “The Esterel v5 Language Primer”, 2000.

[7] Frederick Kuhl, Richard Weatherly and Judith Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, Prentice Hall, 1999.