Integrating Realistic Simulation Engines within the MORSE Framework

Arnaud Degroote*, Pierrick Koch†‡, Simon Lacroix†§
*Institut Supérieur de l'Aéronautique et de l'Espace, 31055 Toulouse, France
arnaud.degroote at isae.fr
†CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
‡Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
pierrick.koch at laas.fr
§Univ de Toulouse, LAAS, F-31400 Toulouse, France
simon.lacroix at laas.fr

I. Introduction

As complex systems, robots integrate a variety of sub-systems that rely on a large spectra of physical processes. Dynamics is of course the primal concern, and it varies a lot depending on the environment and kind of robots considered: there is not much to compare between rigid-body mechanics, fluid dynamics or wheel-soil interactions for instance. Perception implies optics, electromagnetism, acoustics, also declined according to the kind of sensors used and the considered environments. Besides these robot-related physical processes, the environment itself is defined by a series of properties and dynamic processes, that either pertain to physics (*e.g.* atmospheric phenomena that may impact flight mechanics or perception) or are related to other actors governed by specific models (*e.g.* crowd dynamics, road traffic, and even humans interacting with the robots).

Robotics simulators developed within the robotics community are of course far from integrating this whole spectrum of processes and models. Their design is mostly driven by other concerns, such as real-time property, compliance with the software architecture within which the functions to evaluate are integrated, ease of deployment and use... They are mostly used to test, evaluate or validate the *integration* of a series of functions, and the realism of the simulated processes is often not an important concern.

While such simulators are very beneficial to robotics developments, their lack of realism does not properly fill the gap between simulations and actual tests: there is a growing interest in exploiting more realistic models within robotics simulators. The literature abounds with *specialized simulators*, that focus on a given physics phenomenon: the issue of integrating such simulators within a robotics simulator, in a composable and reusable manner, pertains to the simulator *architecture*. This paper presents a way to tackle this integration issue, based on the robotics simulator Morse [11, 3] and the High Level Architecture standard (HLA, [12]).

II. Robotics Simulators

Robotics simulators such as Gazebo [2], V-REP [7], or Morse [11, 3] rely more or less on the same architecture: they are built upon one (or more) classic physics engine such as ODE [4] or Bullet [1], and a graphical engine to edit the environment and simulate vision and depth sensors. Simulated sensors and controllers are embedded in a component / plug-in system. All these components run in the same process, using multiples threads for parallel computations [1]. While such a design allows to test in real time the integration of different robotics components and rather "high-level" algorithms, it fails to scale along two dimensions:

- The increase in the number of simulated robots;
- The augmentation of the realism of the simulated sensors, actuators and environments.

Besides CPU/GPU performance limitations, integrating an existing specialized simulation may be difficult in practice, as existing simulators usually do not provide the interfaces defined by the component / plug-in system. Last, even if several physics engines are supported, only one is supported for a given simulation run, as these simulators are configured at start-up with one physics engine. This is an issue for fine-grained simulation of teams of heterogeneous robots, that may include aerial, marine and ground robots.

A. Morse design

The Morse simulator relies on the open-source modeller Blender and the physical engine Bullet. This tight integration within Blender allows to easily construct realistic robots and environments. Blender being fully scriptable in Python, one can import scenes from third party environment models, *e.g.* multi-layered terrain maps provided by geographic agencies or companies.

As it is usual for robotics simulators, Morse provides a components system, allowing to easily integrate new sensors and actuators on-board the simulated robot. But Morse goes further in the component decomposition and the separation of concerns, by providing modifiers and datastream handlers. Modifiers are pieces of code allowing to change slightly the behaviour of a sensor, from simple data type conversion (Euler to quaternion angle representations, units) to more complex

[1]In Gazebo, there is in fact two processes, a viewer allowing interactions with the simulation *gzclient* and the simulator itself *gzserver* following the described architecture

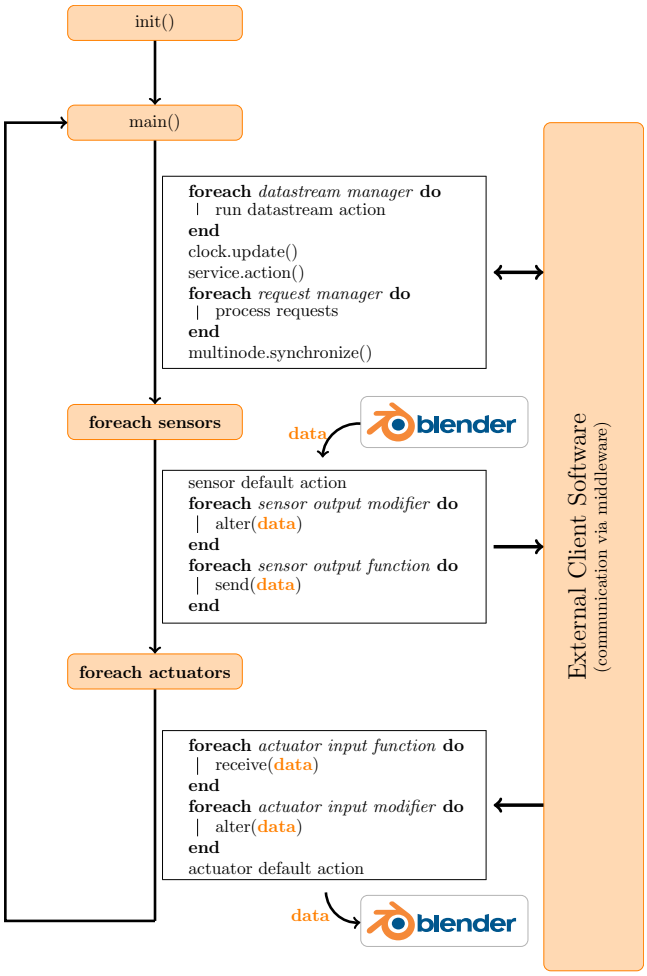

Fig. 1. Morse main loop overview

noise models. Datastream handlers adapts the data to robotics middleware specific format, for both data-oriented interfaces and service-oriented interfaces. This feature allows to transparently use Morse with a variety of robotics architectures, such as GeNoM-based, Orocos-YARP, Orocos-ROS, standard ROS, MOOS... Robots running different architectures can hence be jointly simulated – which proved to significantly ease multiple partners integration in collaborative projects. Figure 1 illustrates how the different Morse components interact during one simulation loop.

Simulation scenes, including the components to use, their configurations and interactions are described using the Builder API, an internal domain-specific language based on Python. It hides the Blender complexity to users, but also allows to dynamically program a scene, thus easing robustness tests of algorithms in various situations.

III. Distributed simulations

To augment the level of realism of a robotics simulator using additional specialized simulators, or to augment the number of robots involved in a simulation, the only way is to distribute the simulation processes over a network of CPUs. This raises the issue of *synchronisation*, as the simulators deployed on the various CPUs have their own computation requirements and scheduler, or the CPUs may differ – not to mention the delays induced by the network. Besides, one need tools to allow the simulators to exchange information between the simulators.

A. High Level Architecture (HLA)

Distributed simulations are largely used in more mature industries, such as space [15] or aeronautic [10, 9] industries, because they allow to make precise and realistic simulations. This is made possible thanks to the High Level Architecture standard, which defines solutions to the issues of information exchanges and time management between the simulators.

HLA is an open international standard, developed by the Simulation Interoperability Standards Organization (SISO) and published by IEEE. In the HLA terminology, a *federate* is an HLA compliant simulator, while a *federation* is the set of simulators connected for one distributed simulation. Without entering too much in the details, HLA defines an API that allows to:

- model the content of one simulator (the Simulation Object Model or SOM), and what is exchanged in the federation (the Federation Object Model or FOM) (*i.e.* the set of objects exchanged in a given simulation);
- manage the federation itself (simulator entering or going out the federation), and the objects that each federate manages;
- provide several time management policies.

Various implementations of this standard are available (*e.g.* [5, 6, 14]): these *RunTime Infrastructures* can be viewed as simulation middlewares. One of the great benefits of using HLA is that a wide variety of specific simulators are compliant with the interfaces it defines. In robotics, there has been a few attempts to develop distributed simulation suites using HLA [16, 17, 13], but none became widely used yet.

B. Morse and HLA

To handle large number of robots in a simulation, Morse proposes a multi-node mode. Basically, each node handles a small subset of robots and simulates their sensors and actuators, and synchronizes the robot positions[2], so that all simulated robots are known to all nodes (which allows sensors to perceive robots from other nodes for instance). The same mechanism is also used for hybrid simulation, *i.e.* in which real robots interact with simulated robots – this specific case imposing that the simulation runs in real-time. A simple in-house protocol has been defined for this purpose, but it yields very primitive distributed simulations, only deployable on homogeneous nodes.

To exploit specialized realistic simulators and develop more realistic multi-robot simulations, we extended the Morse architecture to comply with the HLA specifications. As a use case, we present here the integration of a JSBSim [8], a flight simulator that exploits an accurate Flight Dynamic Model (FDM) engine (in its current implementation, Morse has indeed only a simplistic flight model for quadrotors or

[2]Hence the call to `multinode.synchronize()` in the `main()` of the Morse loop shown figure 1.

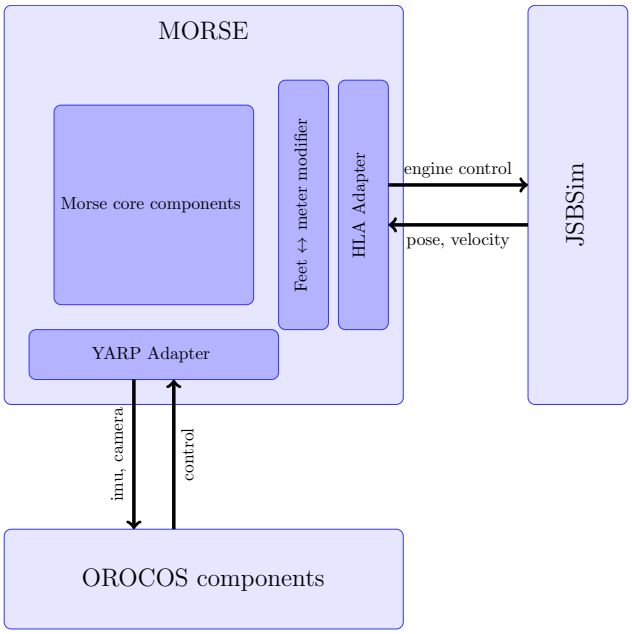

Fig. 2. An Orocos-based robot simulation with Morse and JSBSim

helicopters, good enough for testing integration and high-level algorithms, but not precise enough to test flight control laws). In this particular case, it is possible to use a direct integration of the JSBSim engine into a Morse component, but such an approach would not scale over the long-term (and such an integration is not always possible, depending on the specialized simulator architecture). To achieve the integration of Morse and JSBSim using HLA, nothing had to be changed in the Morse core architecture: we only developed a new datastream handler for HLA, allowing to retrieve or send arbitrary data from Morse using the HLA API. On the other side, we developed a simple C++ HLA wrapper around JSBSim, using the CERTI [14] HLA implementation.

Figure 2 depicts the interactions between an Orocos software architecture with a Morse/JSBSim integrated simulator. Orocos components communicates with the Morse simulator using the YARP middleware, while the Morse core interacts through HLA with the JSBSim node, sending commands (coming directly from Orocos, or after some computation of a Morse actuator), and receiving pose and velocity updates. These values can then be used to simulate on-board sensors output (*e.g.* IMU). Note the presence of feet/meter modifier, to comply with the units used in JSBSim (this conversion could have been made within the C++ federate, but since most FDM use imperial units, we adapted the Morse interface).

Thanks to HLA, a multi-node simulation is now only a specific instantiation of an HLA architecture with various Morse HLA nodes. Not only this standardizes and eases the deployments of large multi-robot simulations, but the synchronization of the nodes are now guaranteed.

IV. Summary

We briefly described how the Morse architecture has been adapted to interact in a HLA federation. This is work in progress, and the developments are still preliminary (the source code is on https://github.com/adegroote/morse-jsbsim). Future work will consist in extending the realism of simulations by integrating additional specialized simulators (*e.g.* an atmosphere simulator for autonomous soaring developments), and on the user-interface, which should ease the definition of distributed simulations. Among the longer term issues that remain to be tackled, the two following are worth to study:

- The definition of simulation descriptions in order to provide generic interfaces and interoperability between simulators. Indeed, while HLA defines a grammar, it does not define the vocabulary exchanged between federates. To provide real interoperability between simulators, this vocabulary (the Federation Object Management) must be generic and descriptive enough. This model problem is still open, and should be discussed within the community.
- The maintenance of the consistency of the environment models that are distributed over an HLA federation: *e.g.* when ruts are caused by robot traverses on a terramechanics terrain model, the terrain appearance must be updated accordingly for the vision / Lidar simulators. HLA could handle such changes, but their modeling is a tough issue.

References

[1] Bullet website. URL http://bulletphysics.org/wordpress/.
[2] Gazebo website. URL http://gazebosim.org/.
[3] Morse website. URL https://morse.openrobots.org.
[4] ODE website. URL http://ode.org/.
[5] Open RTI source code. URL http://sourceforge.net/projects/openrti/.
[6] Portico HLA RTI. URL www.porticoproject.or.
[7] V-REP website. URL http://www.coppeliarobotics.com/.
[8] Jon S Berndt. JSBSim: An open source flight dynamics model. In *in C++. AIAA*. Citeseer, 2004.
[9] W. Bo, L. Hu, Z. Yibo, and S. Yijie. HLA based collaborative simulation of civil aircraft ground services. In *Information Computing and Applications*, pages 734–741. Springer, 2011.
[10] J-B. Chaudron, D. Saussié, P. Siron, and M. Adelantado. Real-time aircraft simulation using HLA standard. In *IEEE AESS Simulation in Aerospace 2011, Toulouse (France)*, June 2011.
[11] G. Echeverria, S. Lemaignan, A. Degroote, S. Lacroix, M. Karg, P. Koch, C. Lesire, and S. Stinckwich. Simulating Complex Robotic Scenarios with MORSE. In *SIMPAR*, 2012.
[12] Frederick Kuhl, Richard Weatherly, and Judith Dahmann. *Creating computer simulation systems: an introduction to the high level architecture*. Prentice Hall PTR, 1999.
[13] Patricio Nebot, Joaqunf Torres-Sospedra, and Rafael J. Martnez. A New HLA-Based Distributed Control Architecture for Agricultural Teams of Robots in Hybrid Applications with Real and Simulated Devices or Environments. *Sensors*, 11(4):4395–4400, April 2001.
[14] E. Noulard, J-Y. Rousselot, and P. Siron. CERTI, an Open Source RTI, why and how. In *Spring Simulation Interoperability Workshop*, pages 23–27, 2009.
[15] Michael R Reid and Edward I Powers. An evaluation of the high level architecture (HLA) as a framework for NASA modeling and simulation. In *25th NASA Software Engineering Workshop, Goddard Space Flight Center, Greenbelt (USA)*, 2000.
[16] L. Winkler and H. Wörn. Symbricator3D – a distributed simulation environment for modular robots. In *Intelligent Robotics and Applications*. Springer, 2009.
[17] L. Xiang, L. Xunbo, and C. Liang. Multi-disciplinary modeling and collaborative simulation of multi-robot systems based on HLA. In *IEEE International Conference on Robotics and Biomimetics*, Dec 2007.