

# A new simulation framework for autonomy in robotic missions

Lorenzo Flückiger, Christian Neukom

*QSS Group Inc., NASA Ames Research Center, {lorenzo,cneukom}@email.arc.nasa.gov*

## Abstract

*Autonomy is a key enabling factor in the advancement of robotics for remote exploration. As the level of resources devoted to this effort continues to increase, it has become equally important to provide simulation tools and environments to scientists in which to test the autonomy algorithms. While industrial robotics benefits from a variety of high quality simulation tools, researchers developing autonomy software are still dependent primarily on block-world simulations. The Mission Simulation Facility (MSF) project addresses this shortcoming by providing a simulation toolkit that will enable developers of autonomous control systems to test their systems' performance against integrated, standardized simulations of NASA mission scenarios. The MSF provides a distributed architecture that connects the autonomous system to a set of simulated components replacing the robot hardware and its environment.*

## 1 Introduction

The exploration of remote environments using robotic systems is a very challenging task. The physical robot not only has to withstand the hostile elements of the environment, it also needs to demonstrate high levels of autonomy to accomplish its tasks without continuous human control or intervention.

The typical human-robot interaction scenario for an exploration rover on Mars is strongly affected by communication issues: scientists on earth can only communicate with the Martian rover once a day, with a limited bandwidth and for a period of only several hours. Communication becomes an even greater issue for planets farther away from Earth such as Europa. On Earth, the exploration of remote sites (for example the ocean under the Arctic ice) faces similar problems. Therefore, besides the development of ground tools to help scientists plan, test and visualize remote science experiments, the rover needs autonomy to perform tasks without direct human control. To accomplish critical science missions, the rover needs to be able to explore an unstructured world and make decisions based on on-board sensors. In addition, the rover needs to be capable of adapting its mission to unanticipated events in order to maximize the science return and ensure rover safety [10].

Research in autonomy for robotic platforms used in remote exploration is therefore critical for mission success. However, this research area faces several challenges when it comes to testing new autonomy concepts:

- Custom simulators usually provide testbeds for individual algorithms without the context of an integrated robotic mission.
- Access to hardware is either very costly or not available at all. Even when hardware is available, time and resource constraints often limit test scenarios to just a few environments. In addition, it can be difficult to control the parameters of the test, making the experiments hard to repeat.

To address these shortcomings the Mission Simulation Facility (MSF) project was initiated in early 2001 to support autonomy research for robotic systems.

### 1.1 MSF Goals

The goal of the MSF is to develop a simulation framework and suite of simulation tools to support research in autonomy for remote exploration. Such a system will allow developers of autonomy software to test their models in a high-fidelity simulation and evaluate their system's performance against a set of integrated, standardized simulations. There is currently a large gap between autonomy software at the research level and software that is ready for mission insertion. It is our vision that the MSF will bridge this gap by providing researchers with high-fidelity simulations of mission scenarios to test their software in realistic, complex environments.

The research community in autonomous systems can not rely on commercial tools for simulation because these are not applicable to robots with various instruments evolving in unstructured environments. Thus too often, each research lab has to develop its own simulator. Such simulators are usually oriented towards "block-world" models (answering most needs while keeping simplicity). For example, the simulator embedded in the Saphira architecture [4] is useful to experimenting with robotic autonomy, but is not suitable for planetary missions due to its 2D world model and limited available sensors. On the other hand, high quality simulators are developed for specific problems such as robot dynamics [8, 7, 11] or instrument and environment modeling [6, 3, 9]. These

tools offer highly accurate models but are oriented towards engineering design or mission-ready simulation. Consequently, even though many high fidelity models exist, they are difficult to combine into an integrated simulation. Typically these tools are tied to a specific operating system or computer language and are not designed for applications outside of their nominal scope. The MSF is designed to connect these models through the Mission Simulation Toolkit (MST), a software package comprising 1) a framework for connecting and synchronizing distributed software models, 2) generic interfaces abstracted from the transport layer, and 3) a set of basic components required for a simulation.

## 1.2 Scope and Applications

The MSF architecture is designed to support multiple mission platforms (e.g. planetary robots, spacecraft, underwater vehicles), with the initial focus on supporting planetary rovers. The components needed for a rover simulation would include a terrain model, site information (such as coloration and mineral composition), an environment model (sun position, lighting, temperature, etc), a rover including a kinematic model and on-board sensors, and several scientific instruments. It is up to the user to decide what granularity of models best suits the purpose of his simulation. For example, a user who is concerned primarily with collecting scientific data may not require a sophisticated rover model because he may not care how the rover gets from one point of interest to another. On the other hand, a user who is developing a trajectory generator may want to control the individual wheels of a rover. For this reason the MSF provides both high and low level interfaces to a number of standard models. This dual level interface is also needed to support a variety of robot architectures. Since very different approaches currently exist (see [1] for references), the MSF has to provide an interface generic enough to equally support hierarchical, behavioral or hybrid robot architectures. Figure 1 shows the concepts of the MSF distributed simulation relying on a common communication framework to connect models and autonomy software.

## 2 Proposed Framework

The realization of the goals mentioned in the previous section requires the construction of the following components:

- A distributed architecture** that implements a communication layer.
- A set of models** allowing the simulation of robots, environment and science instruments.
- Scenario representation** through the description of robot systems and their environment.
- Databases** used both as a source of data for the model and as storage for the result of the simulation.

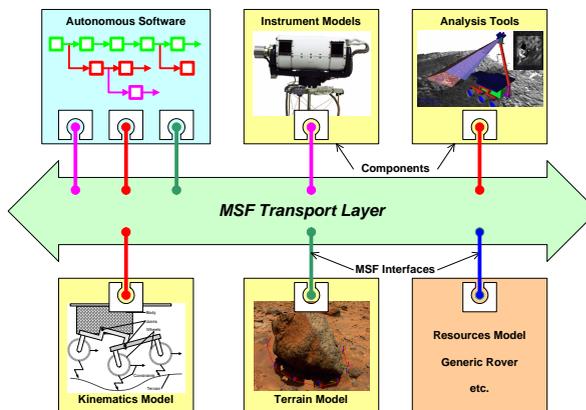


Figure 1: Overview of the Mission Simulation Facility.

These components (except databases which are not yet treated by the MSF team) are described in the following sections.

### 2.1 Distributed Architecture

The MSF architecture is derived from two main requirements: to support distributed simulation on multiple platforms and to ensure extensibility through an open architecture.

**Multiple Platform Support.** Users of the MSF (autonomy developers) typically develop their tools in a variety of environments, for example, a LISP program under Solaris on a Sun workstation or a C++ program under Linux on an Intel PC. Target systems for the autonomy software (the rover control software) may also be developed on different operating systems and hardware platforms. Such software could for instance rely on a particular flavor of Linux running on a PC-104 Pentium board, or could be a dedicated embedded system running VxWorks. The MSF project does not intend to develop all the simulation components but rather will take advantage of existing tools. To minimize the adaptation requirements, each particular software component should be usable by the MSF on the original platform for which it was developed.

**Open architecture.** The MSF is a general purpose testbed for mission simulation rather than a specific simulator, which implies that different sets of components will be used for different scenarios or different domains. For instance, one might use a kinematics model for a rover and a fluid dynamics model for an underwater vehicle. In addition, a component should be usable in multiple scenarios, which means that the same rover kinematics model could be used for various rovers with particular payloads operating in different environments. Finally, it should be possible to replace a component of a specific type with another that performs the same function but at a different level of fidelity. A simple kinematics rover simulator could be adequate to test high level autonomy con-

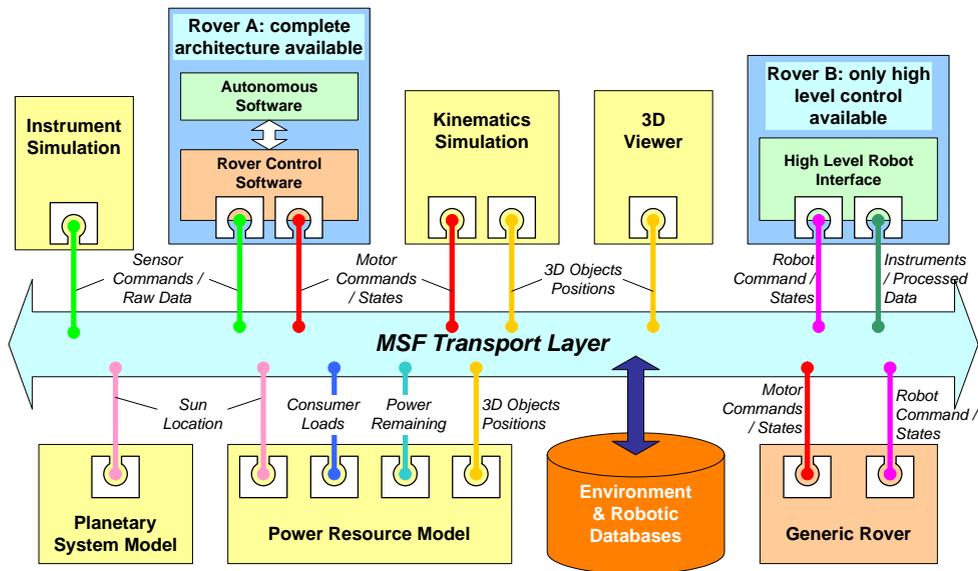


Figure 2: MSF simulation with several components communicating through common interfaces.

cepts such as path planning, while a dynamics rover simulator including accurate soil-wheel interactions may be required to test an autonomous control system for the mobility of the rover<sup>1</sup>. It is therefore essential that the MSF define clear interfaces between the components to facilitate the exchange of components included in the MSF toolkit with those developed at other research institutions.

A way to satisfy the above requirements is to have a distributed architecture where components communicate with each other using a common transport layer. The MSF is built on top of the standardized High Level Architecture (HLA)<sup>2</sup> [5] which is an architecture for simulation reuse and inter-operability developed by the Defense Modeling and Simulation Office (DMSO). The MSF currently uses the Runtime Infrastructure (RTI), a software implementation of HLA, freely distributed by DMSO. The HLA/RTI provides the MSF the following services:

- Multi-platform support: IRIX, Solaris, Linux, Win32 and VxWorks
- C++ and Java bindings
- Choice of transport protocol: TCP (reliable) or UDP (fast)
- Publish/Subscribe scheme
- Communication through objects or messages
- Various time management schemes for simulation synchronization

To facilitate the integration of components in an MSF-based simulation, an abstraction layer has been developed on top of the HLA, the Federate ToolKit (FTK). FTK is

<sup>1</sup>This latter case is however not directly addressed by the MSF in its first phase

<sup>2</sup>The HLA was approved as an open standard through the IEEE Standard 1516 in September 2000. See Defense Modeling and Simulation Office website: <http://www.dmsomil/public/transition/hla>.

responsible for the integration of communication entities with the Runtime Infrastructure. The communication objects and messages defining the MSF interface are easily designed using the Unified Modeling Language (UML). All the necessary C++ code to use these communication entities is generated automatically.

Figure 2 gives an example of a MSF-based simulation with several interconnected components. Two separate rover autonomy software executions are participating in the simulation: Rover A software is provided by a lab having a complete rover architecture (and probably a real rover) from the high level control down to the hardware control; Rover B software comes from a lab working only on high level autonomous algorithms and without hardware control. When Rover A software sends commands to its actuators (e.g. `motor1.start(speed, duration)`), the commands are routed to the simulator rather than to real hardware. The Kinematics Simulator accepts such commands and computes the behavior of the rover on the terrain regarding these inputs. When Rover B issues a high level command to its base controller (e.g. `roverB.moveto(position, obstacle-avoidance=on)`), the Generic Rover model catches this message and produces motor commands for a simple model of a rover causing it to move from one location to another while avoiding obstacles. These motor commands can be processed by the same Kinematics Simulator that is used for controlling Rover A. The same type of scheme is used when controlling instruments, generalized as sensors in Figure 2 (the figure does not show the full path of information flow). In addition to the Kinematics and Instruments models, a Power Resource model is participating in the simulation. It monitors the load of each actuator as well as the power generated by solar panels

and computes the power remaining in the rover’s batteries. The power output of the solar panels depends on the orientation of the solar panels relative to the sun. The position of the panels is provided by the kinematics model and the sun’s position is delivered by another component computing the Ephemerides. This example shows how different components are reusable for different scenarios, and how the definitions of the networked MSF interfaces removes all the dependencies between the components.

## 2.2 Simulation Models

The MSF makes it possible to replace robotics hardware and environments with simulated components, which implies having a library of models of the robots themselves and of various environments representative of future missions. The interaction of a rover model with its environment involves several elements. For example, when the robot moves to a new location, the motors drive the wheels, which in turn move over the terrain according to the forces of gravity and shape of the terrain. The robot obtains engineering data through its sensors, while on-board science instruments collect data from the virtual environment. Robotic instruments such as a rock grinder may also be used to interact with the environment.

The models for a simulation can be developed at very different levels of sophistication. For instance, the model of a rover driving over a terrain could work with the assumption of a flat surface and perfect motion, or could involve a full dynamic system including wheel slippage. The following paragraph briefly presents different approaches.

**Level of simulation.** The purpose of the science robotic system is to explore its surroundings by actively taking measurements. Figure 3 depicts a number of modules composing a rover software architecture with flows of two types of information: actuation (output) and sensing (input). In this simplified example, the Goal Decomposer on the Rover Autonomy side hands tasks to a Path Planner, which in turn interacts with a Locomotion Controller. The latter communicates with the rover hardware, sending commands to actuators and obtaining proprioceptive sensor data. On the side of the figure labeled “Science Autonomy”, the world is sensed using a camera controlled by a Vision System that feeds an Image Interpreter module. The latter supplies information to high level World Analysis algorithms. In actual autonomous systems, the flow of commands and sensory inputs are closely connected. For example, the obstacle avoidance algorithm takes actions based on the information provided by the obstacle detection algorithm. While there is a fair amount of cross-over of information between the science and rover autonomy, generally more sensory input data is directed to the science autonomy side and much of the output is concerned with the functioning of the Rover Autonomy. The simulator could be attached at any of these levels of the rover architecture on

each side, and not necessarily at the same level for each area of specialization.

It is interesting to note that in general it is easier to build a simulator that connects at the low level of the “actuation flow” and at the high level of the “sensing flow”. On the “actuation flow” side, building a simulator that sends low level commands to the actuators is easier because: 1) access to hardware is less dependent of the rover software architecture and 2) the simulator does not need to replace some behaviors of the rover (e.g. obstacle avoidance in the example above). In contrast, a simulator could more easily feed the high level modules of “sensing flow because”: 1) the simulator knows the truth model and thus can directly provide high level information about the scene (e.g. there is a big rock in front of the rover) and 2) an accurate instrument model that takes into account physical laws (illumination, material texture, etc) is difficult to build.

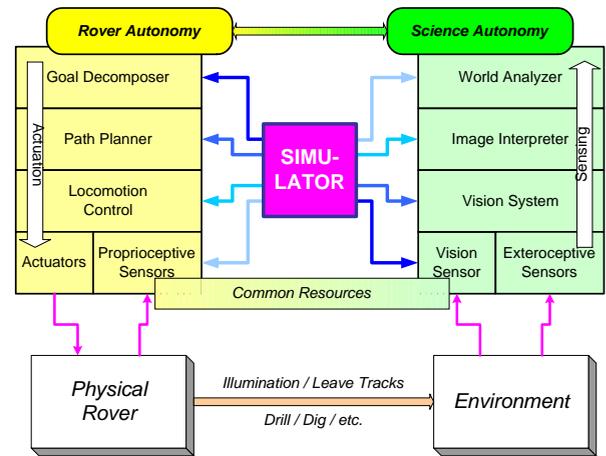


Figure 3: Various levels of simulation possible for a hierarchical robot software architecture.

Because the MSF is being built incrementally, our initial focus is on creating a strong infrastructure and providing a toolkit that includes only a few simple models. Once the infrastructure is in place, new models of higher fidelity or models that address different domains will be added. Currently the MSF depends on an existing kinematics simulator for the rover mechanics (controlled by low level inputs such as wheel speed) and feeds control algorithms with high level information coming from the truth model (e.g. “this rock contains carbonates”). Section 3 briefly presents the components used in the current MSF implementation. Ongoing collaborations will provide improved models for terrain generation, dynamics simulation and science instruments.

**Required Components.** The MSF is a general framework for connecting multiple components to build various types of simulations. In this sense, the MSF does not

enforce the use of a particular set of simulators, and lets the user build to meet his own needs. However, it is possible to identify the components required for a minimal planetary simulation:

**Environment Generation.** An environment model is essentially composed of a terrain surface with attached properties. The data can either be generated at runtime by a terrain server, or precomputed and stored in a database. Ideally the data will be generated by terrain models from a set of input parameters (location, resolution, type of terrain, rock density, etc.) but real terrain data can be used as well in certain situations (the MSF currently uses data collected during various Mars mission and Earth field tests). An environment model may also include data representing atmospheric conditions, celestial body positions, lighting, etc.

**Robot Behavior.** From a high level point of view, the most significant representation of the robot model is the mechanical behavior, which determines the positions of the robot and its effectors. The robot model also needs to include other resources including proprioceptive sensors, power, computation and communication. Depending on the level of autonomy being tested, the model may also include generic robot capabilities such as obstacle avoidance. Finally, it is critical for the robot simulation to represent and handle failures because they play an important role in robot autonomy.

**Instrument Models.** Instrument models provide autonomy developers with data from the robot's environment, feeding their algorithms. Again, depending on the level of autonomy being tested, the data products could be the raw data an instrument normally returns (e.g. a photo-realistic image) or pre-processed information (e.g. the response from a spectrometer module: "there is carbonate in this rock"). The MSF will provide models for generic instruments including cameras and spectrometers. Complete instrument packages such as those intended to fly on future missions may be added to the library as the models become available.

**Data Analysis.** Tools for collecting and analyzing data from simulation runs are fundamental to evaluate autonomy software. The HLA communication layer and the tools developed for it provide easy collection of all information exchanged among the MSF components. One of the first qualitative evaluation tools used in the MSF is a 3D viewer allowing the user to observe robot behaviors in a simulated world and to see views from cameras. The same viewer can display abstract object parameters including the torque on motors, the field of view angle of a camera, or the data of an instrument.

## 2.3 Scenario Description

Because the MSF is a general simulation platform, it needs to be easily configurable for different scenarios. In general, a scenario comprises the robotic systems description, the environment description, and some simulation specific parameters. The robot description defines the kinematics (or dynamics) parameters of the robot's mechanical structure, the actuators, various instruments composing the payload<sup>3</sup> and several resource modules (e.g. battery, memory). The environment is composed of the following: 1) a reference to the terrain data that will be used to compute the kinematics of the rover, 2) the data inferred by the instruments, 3) the type of atmosphere and other environmental parameters, 4) the location of the mission (which planet, latitude and longitude) and 5) the epoch of the simulation.

No parameters regarding robots or environment are encoded in any MSF component. Instead, the components obtain their specifications from the scenario description file when a new simulation is initiated. A component defined in this manner will then produce simulated results on the basis of an external description. The scenario description file could in fact be a set of files, each referenced in a main file. Such a setup creates a unique parameter, the file URL, which completely defines a specific scenario in a MSF simulation. The MSF currently uses a file description that is a proprietary extension of the Robot File Format used by VirtualRobot [2], which contains a subset of the capabilities listed above. The MSF plans to use an XML scenario description with dedicated Document Type Definition (DTD). This will provide a description of the simulation that is human readable (and editable) with the bonus of having existing graphical editors to modify the file and the necessary tools to parse it. In addition, tools to exchange XML files over the network are widely available, which is important for a distributed simulation.

To ensure coherency, all the MSF components participating in a simulation will read the same scenario description file. However, each component extracts only the information relevant to its domain. For example, the kinematics simulator will identify all the dimensional parameters of a rover, while the instrument model will only need to access the rover's name. Having a single source for describing the scenario is critical for software configuration management and testing. A seemingly simple change to the model robot such as increasing the power of the motors can affect multiple software components such as the power model, dynamic simulator, abstract reasoning layer or the visual representation. If the scenario description were not limited to a single file, the result-

<sup>3</sup>Proprioceptive sensors, such as the incremental counter of a DC motor, are considered part of the actuator. Low level control between a DC motor and its controller is not simulated at the level of accuracy that is addressable by the MSF.

ing simulation could become unmanageable to verify and distribute.

### 3 Results: a First MSF Instantiation with Three Components

A prototype of the Mission Simulation ToolKit (MST) has been implemented, including a tool for generating object-oriented communication classes in C++ from a UML description. The communication classes rely on an abstraction layer (FTK), hiding the complexity of HLA. The design of the communication infrastructure dramatically facilitates the integration of simulation components within the MSF architecture while satisfying all MSF performance requirements for data transfer and message exchange between components.

We recently applied our framework to the Virtual Intelligent Planetary Exploration Rover (VIPER)[2] and were able to achieve full integration in just a few days. VIPER is a system linking the capabilities of a high fidelity virtual environment (VIZ), a kinematically accurate generic robot simulator (VirtualRobot) and a flexible plan execution module (Conditional Executive). The integrated system allows users to simulate and visualize the behavior of the robot during the execution of task plans under development for a planetary mission. Before our integration effort, the three components of VIPER were linked through various proprietary socket methods. With the introduction of the MSF backbone, the modules depend on a single communication layer by exchanging common objects and messages from a well defined API. Besides improving the coherency and reliability of the system, the MSF made VIPER directly extensible (new components can be plugged in) and reusable (the existing components can be applied to new scenarios).

### 4 Conclusion and Future Directions

This paper has presented the MSF simulation framework for autonomy research that makes it possible to integrate available models of robotics systems and environment with autonomy software. The toolkit provides developers a facile method to create communication entities that define the API between the components participating in a simulation. By collaborating with researchers from both the autonomy and rover control community, the MSF project will define a standardized interface allowing the components of a simulation to be easily exchanged. In addition, the distributed architecture enables researchers to execute their autonomy software on the intended platform while allowing model developers an easy way to connect their components in the simulation.

Currently the MSF provides an initial set of components based on the VIPER system (Executive, Kinematics and Visualization) permitting users rapid creation of a simulation for a particular robot scenario. This library of

components is being expanded to include multi-body dynamics simulation, science instrument models, environment generation and a fully capable generic rover. It is our hope that this library will grow as more users begin to use the MSF and make their models available to other research groups. We are also hopeful that the flexible design and ease of use will make the MSF attractive to a large community of users developing autonomy for robotic missions.

### Acknowledgments

This project is funded by the NASA Intelligent Systems Program. The authors would like to thank the task requester Butler Hine as well as the complete MSF team for their support: Greg Pisanich, Laura Plice and Michael Wagner.

### References

- [1] Ève Coste-Manière and Reids Simmons. Architecture, the backbone of robotic systems. In *Proceedings of the ICRA 2001*, 2001.
- [2] L. Edwards, L. Flückiger, and R. Washington. VIPER: Virtual intelligent planetary exploration rover. In *Proceedings of i-SAIRAS 2001*, 2001.
- [3] R. W. Gaskell, J. B. Collier, L. E. Huseman, and R. L. Chen. Synthetic environments for simulated missions. In *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, March 2001.
- [4] Kurt Konoldige, Karen Myers, Enrique Ruspini, and Alessandro Saffiotti. The Saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, pages 215–235, 1997.
- [5] F. Kuhl, R. Weatherly, and J. Dahmann. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall, 2000.
- [6] M. Lee, R. Weidner, and W. Lu. Mission lifecycle modeling and simulation. In *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, March 2000.
- [7] Patrick Christopher Leger. *Automated Synthesis and Optimization of Robot Configurations: An Evolutionary Approach*. PhD thesis, Pittsburgh, Pennsylvania, 1999.
- [8] S. McMillan. *Computational Dynamics for Robotic Systems on Land and Under Water*. PhD thesis, Ohio State University, Columbus, OH, 1994.
- [9] P. Tompkins, A. Stentz, and W.L. Whittaker. Automated surface mission planning considering terrain, shadows, resources and time. In *Proceedings of i-SAIRAS 2001*, 2001.
- [10] R. Washington, K. Golden, J. Bresina, D. E. Smith, C. Anderson, and T. Smith. Autonomous rovers for Mars exploration. In *Proceedings of The 1999 IEEE Aerospace Conference*, 1999.
- [11] J. Yen, A. Jain, and J. Balaram. ROAMS: Rover Analysis, Modeling and Simulation. In *Proceedings of i-SAIRAS 1999*, pages 249–254, 1999.