EXPERIMENTS WITH AUTONOMOUS SOFTWARE FOR PLANETARY ROBOTS: A SIMULATION SUCCESS STORY

Lorenzo Flückiger[(1)], Christian Neukom[(2)], Greg Pisanich[(3)], Eric Buchanan[(4)], Michael Wagner[(5)], Laura Plice[(6)]

[(1), (2), (3)] QSS Group, Inc., NASA Ames Research Center, Moffett Field, CA-94035, USA
Email: {lorenzo, cneukom, gp}@email.arc.nasa.gov
[(4)] MCT, NASA Ames Research Center, Email: buchanan@email.arc.nasa.gov
[(5)] Jet Propulsion Laboratory (JPL), Pasadena, CA-91109, Email: Michael.D.Wagner@jpl.nasa.gov
[(6)] Formerly QSS Group, Inc., Email: plice@ix.netcom.com

ABSTRACT

Autonomy is a key enabling factor for robotic exploration. There continues to be a large gap between autonomy software (at the research level) and software that is ready for insertion into near-term space missions. The Mission Simulation Facility (MSF) project attempts to bridge this gap by providing a simulation framework and a suite of tools to support research and maturation of autonomy. The MSF has a proven basis in applications for surface planetary missions. Moreover, the innovative framework is readily adaptable to a large range of missions by providing component insertion and a mixed-fidelity simulation.
This paper first describes the requirements for effective autonomy software development and then presents technology developed to this model that resulted in the Mission Simulation Toolkit (MST). It further shows how this toolkit was applied to several projects and the lessons learned during the process. Finally, future directions and applications are presented.

1. INTRODUCTION

NASA Ames' Mission Simulation Facility (MSF) is designed to address a problem often encountered by researchers in autonomy: how to carry out meaningful testing of autonomy software without real-world robotic platforms (costly in budget and time). The Mission Simulation Facility offers a simulated testing environment including robotic vehicles, terrains, sensors, and vehicle subsystems. The initial MSF release targets users researching autonomy for Mars rovers; however, the MSF technology solution is applicable to many other robotic domains. The MSF project has had four major releases of its software since it's beginning in early 2001. The latest release is an Open Source release. The core technology supporting the MSF has been presented in [1], and the identification of the needs for a mission simulation environment supporting autonomy has been described in [2]. This paper describes how the MSF has successfully supported several research projects in robotic autonomy.

The requirements for a simulator that supports autonomy are significantly different than those of conventional robot simulators. First, since high-level autonomy software is intended to control a complete robotic system, a variety of models (terrain, kinematics, dynamics, sensors, power, electromechanical subsystems, etc.) are required. Second, in order to test autonomy software in a range of situations, it is necessary to provide controllable variability and failure injection into these models. Third, because autonomy software is generally developed on a wide variety of platforms and may target robotic systems that are still under development, support for mixed operating system and flexible interfaces are also needed.

The MSF addresses this particular set of requirements through the use of a distributed framework based on the High Level Architecture (HLA) standard [3]. A key feature of the MSF framework is the ability to plug in new models to replace existing ones with the same services. This enables significant simulation flexibility, particularly the mixing and control of fidelity level. The MSF toolset also provides: automatic code generation from robot interfaces defined with the Unified Modeling Language (UML), methods for maintaining synchronization across distributed simulation systems, XML-based robot description, and an environment server. Finally, the MSF supports a number of third-party products including dynamic models and terrain databases.

To fully illustrate the capabilities and flexibility of the MSF, several experiments that involved testing of autonomous software using MSF simulators are presented. Results and lessons learned from these applications of MSF are then discussed.

2. REQUIREMENTS ANALYSIS

The MSF requires specific features to support robot autonomy in addition to typical requirements of conventional mechanical systems simulators, like response time, model accuracy or data logging.

To illustrate the requirements analysis, a simple scenario depicted in Fig. 1 will be considered. In this scenario a rover is controlled by a Conditional Executive (CE) that is capable of executing plans containing floating branches triggered under specific conditions. The rover has been given a path for a long traverse, and during its drive it will acquire images of the environment. An on-board science processing software analyzes the images, identifies and classifies rocks for scientific priorities. If interesting rocks ("layered" in our example) are detected by the Image Analysis (IA) module, the rover will perform a detour to analyze the rock in more detail. The CE maximizes science return while trying to minimize the use of resources such as battery power and time.

2.1. Multi-model with variable fidelity

Autonomous software is intended to control complex systems comprising multiple resources in unstructured environments. Thus, a simulator replacing some robotic hardware interacting with its environments needs to model a range of behaviors. In engineering, most simulators focus on a specific domain such as multi-body dynamics or modeling a camera with sophisticated algorithms and detailed models. In the MSF context, the models can often be simplified because the model outputs will be interpreted by autonomous software that reasons at a fairly high level of abstraction.

In this example scenario where only the Conditional Executive is tested against the simulator, there is no need to generate photorealistic images of the terrain. The CE only uses the high level information produced by the IA: does the rock contain layers or not? Another aspect of the level of fidelity of the simulation comes from how much of the real rover's control software is tested. Fig. 1 shows an application where the real RC is sending commands to the simulator. Often, to test autonomy, the rover control software is not needed or not available. For this reason, MSF provides some common modules like a "locomotor" (see Section 3.3) to emulate some robot functions. In this scenario, the simulator boundaries will extend to the dashed line in Fig. 1.

The principal models required to support a full mission simulation fall in the following categories:

<u>Environment models</u> for planetary simulation consist essentially of terrain data. However, terrain shape, which is usually represented in the form of an elevation map, is not sufficient and additional layers of information like rock properties need to be included. Various sources of terrain environments have been used in MSF. A common database interface to access the various types of environments has been created jointly with the Jet Propulsion Laboratory (JPL). Real terrain data is useful when the simulator is used during a field test for validation purpose (see Fig. 7). However, such data are often incomplete (holes in the 3D mesh) and lack high level type of information (only terrain texture is available). Synthetic terrains are therefore more useful in creating interesting scenarios for autonomy, and can offer continuous terrain patches over large distances.

<u>Robot movement models</u> compute the robot posture according to the commands received by the actuators and the external environment (terrain) the robot interacts with. The posture is important for analyzing safety constraints and for locating the position and orientation of the instruments in the world. For current planetary simulations, robot kinematics is sufficient because of the very slow speed of today's rovers. However, dynamic models are also useful because failures due to collision with a rock can be simulated.

<u>Actuator and Sensor models</u> allow control of the simulated robot and feed the autonomous component with simulated values. Actuators are only modeled at a high level. Internal models of the electromechanical system or controller loop are not implemented because it usually does not affect the autonomous software. The sensors simulated are mostly proprioceptive sensors that are required to produce correct robot control.

<u>Instrument models</u> allow the autonomous software to respond in mission like situations. However, as mentioned above, instruments can often be modeled as a combination of the hardware sensor and the data analysis software that generates the final, high-level output. Instruments are mainly exteroceptive sensors used for science analysis and are therefore modeled as black boxes when science algorithms are not tested.

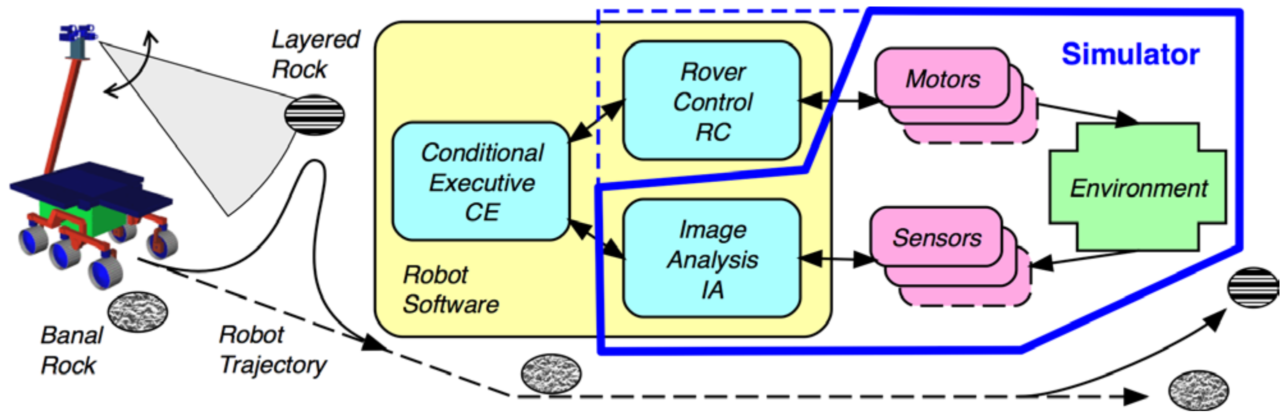

Fig. 1. Illustration of a simple scenario and component diagram showing the boundaries of the simulator

Resource models are key components in the simulation, as one of the critical tasks of the autonomous software is to compute intelligent resource usage. Common resources are power and time, but other resources like memory usage, communication window and bandwidth are also considered for full mission simulations. It should be noted that unlike all the other models, the time model is not a component by itself, but a constraint operating in all the components.

In addition to the multiple models required to simulate a mission scenario, the MSF framework – by design – supports mixed fidelity models in a simulation and allows the replacement of low fidelity models with higher fidelity models when required for specific scenarios.

2.2. Variability and uncertainty injection

To harness the power of the autonomous software being tested, the simulator needs to be able to generate variations in the scenario. In our example, it could be variable time and energy resources that influence the drive to a given point. The capability to inject failures is important for evaluating how the autonomous software will respond to them. A failure condition, in the example scenario, could involve the imager not being able to return data about the presence of a rock, or a stuck wheel.

2.3. Multi-platform and flexible interfaces

Another requirement that is specific to the MSF is to provide a tool that can be used in a research environment. Unlike a simulator that would be developed for testing a specific subsystem before the flight phase, the MSF supports research in autonomy where no computing platform requirements can be imposed. In addition, the scenarios needed for the simulator are often not well defined, thus requiring the MSF to adapt quickly to changing needs through easily reconfigurable interfaces.

3. TECHNOLOGY DEVELOPED

The MSF addresses this unique set of requirements through the use of a distributed framework composed of three elements: the communication layer, a set of interfaces, and a suite of components implementing the basic models required for a typical simulation scenario.

3.1. Communication Layer

The MSF is based on the High Level Architecture (HLA) standard designed for simulations reuse and interoperability. The HLA implementation provides a publish-subscribe scheme with services to time-synchronize components. The MSF has developed a layer on top of HLA, which abstracts several HLA services and automatically ensures the enforcement of HLA rules. This additional layer simplifies the integration of new components into MSF simulations while helping component developers to observe all the HLA conventions without having to master all the HLA intrinsics. This approach allows new models to be plugged in to replace existing ones with the same services, thus providing significant simulation flexibility, particularly in the mixing and control of fidelity level.

3.2. Interfaces

The HLA methods of describing interfaces were insufficient (for fully object oriented paradigms or strong variable typing needs) for supporting the MSF requirements, therefore a process to generate component interfaces has been put in place. The components composing a simulator are using MSF's Communication Objects (CO) to interact with each other. Components see CO's as if they were local objects, and can set their variables or call methods on them. CO's are defined using the Unified Model Language (UML). From the UML model, implementation classes are generated, that internally use HLA objects and interactions. The use of automatic code generation reduces the risk of coding errors, offers great flexibility, and allows rapid creation of a new set of communication objects or extension of an existing one.

3.3. Components

The charter of the MSF was not to develop high fidelity models, but rather to integrate existing ones in its framework. However, the MSF does include a suite of basic components that are essential for creating a simulator. These components are interchangeable with high fidelity models when needed. Below is a description of the core models used in most simulations.

The Simulation Controller manages the startup of all the participating components of a simulation and gives the user control of the simulation (start, pause, stop, reset, quit).

The Mission Simulation Dynamics Engine (MSDE) is a dynamic simulator based on the Open Dynamics Engine (ODE) software. MSDE simulates the interaction of the rover with the terrain during rover movements.

The Locomotor is a generic model that translates high-level robot movement commands into wheel motor speed. The Locomotor works for all wheeled rovers that are supported by the XML robot description file (e.g. no steered wheels, all wheel steering).

A generic power model computes the load put on the battery by the various motors, sensors, and instruments. The model includes a GUI that allows a user to set the battery capacity and obtain readings of the load and battery charge.

HLAB is both a debugging/analysis tool and a command logger. Users can display state variables of all Communication Objects in the simulation and execute available methods on them. During simulation, HLAB acts as the command logger, saving each time-stamped interaction.

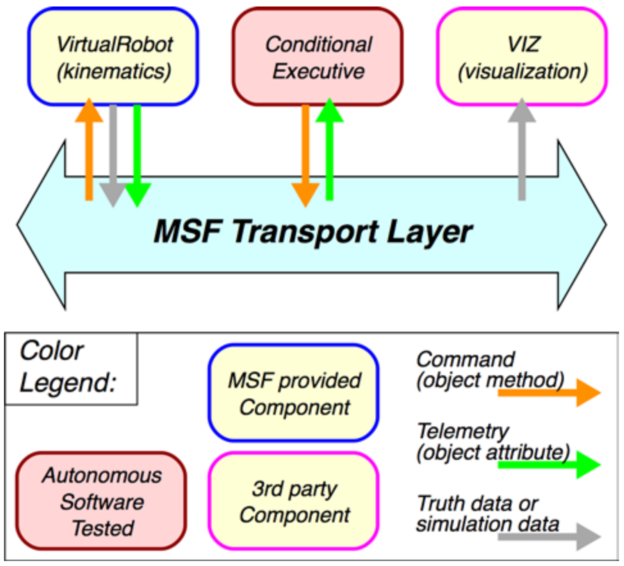

Fig. 2. Simulator configuration for VIPER

Viz is a powerful real-time 3D visualization software developed at Ames by the Intelligent Robotics Group [4]. Viz provides scene graph management, realistic 3D rendering with lighting model and shadow casting using real ephemerides. The MSF has created a plug-in for Viz, allowing it to connect to the simulation, and display any 3D object participating in the simulation.

Most MSF components are generic simulators that can be configured through parameters. For this purpose a robot descriptor has been designed using an XML file format. MSF XML robot files contain information about the various subsystems of the robot (battery, instruments, etc.), its physical configuration (kinematics) and properties (like maximum speed). Thus, every component can extract the required information for the simulated robot from the same common file description.

4. APPLICATIONS

The MST has been continuously utilized by several projects inside NASA Ames since its first release. Three factors have influenced the maturation of MSF: 1) the MSF roadmap to develop a functional simulator for autonomy support, 2) the feedback from the users using the MSF to test their applications and 3) the integration of additional components (internally developed or externally provided). Due to the frequent interaction with its users, and the dynamics of the research environment, the MSF software continuously evolved to support new problems. The MSF software is not a single simulator, but a framework permitting the construction of a variety of simulators. The MSF communication layer remains the same for any application. The MSF interfaces do not require modifications if the application domain does not change (planetary rover), but the interfaces could be extended for a specific application. MSF simulators are tailored for a particular application by combining a set of components that implement the required models. Some of the components are common to most simulations, while others have been developed for a specific purpose. This section presents three different configurations representative of MSF evolution, with an increasing number of components providing more complex mission simulation.

4.1. VIPER

The goal of the Virtual Intelligent Planetary Rover Exploration (VIPER) project was to provide users with a virtual environment in which they could visualize rover behavior and possible execution outcomes of a plan under construction. The initial VIPER system presented in [5] used ad-hoc communication between the components. VIPER was later updated to use the first release of the MSF framework. The system was composed from three components shown in Fig. 2, linked by the MSF framework with minimum customization: conditional plan execution, rover behavior simulation, and 3D visualization.

Plan execution occurred through a Conditional Executive that made decisions based on rules expressed in the Contingent Rover Language (CRL) [6]. The VirtualRobot kinematics model can generate inverse and direct kinematics for any robot structure (described in a configuration file), and was used to compute the rover behavior on a synthetic terrain. Visualization of the rover interacting with the terrain occurred in Viz. Due to the flexibility of MSF, integration of the three components to create VIPER was achieved in a very short time. Fig. 3 shows a scenario where the Conditional Executive was executing floating branches regarding the conditions encountered by the rover.

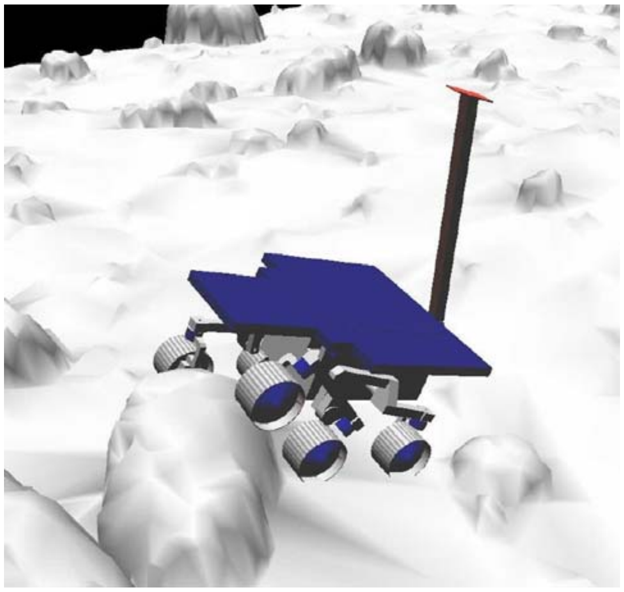

Fig. 3. Simulated K9 rover in a limit configuration driving on a synthetic terrain.

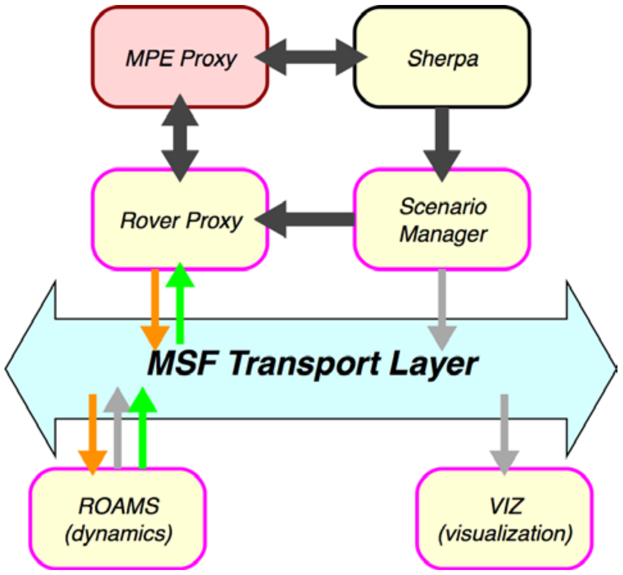

Fig. 4. Software architecture in the REEF-MSF experiment.

4.2. REEF

Apex, developed at Ames, is a reactive, procedure-based planner/scheduler used for mission level task execution [7]. The Apex group has used the MSF to create a prototype of the Requirements Elicitation and Evaluation Facility (REEF) software for a proof of concept demonstration. The objective of this software was to elicit the Mission Planning and Execution system (MPE) behavior requirements from diverse mission experts and then to evaluate the MPE behavior specifications in diverse scenarios. Fig. 4 depicts the combined software architecture. The box labeled "MPE Proxy" is Apex running a simulation of the MPE. The user enters rover behavior through the Apex GUI called Sherpa, and defines the scenario via the scenario manager. The rover proxy is the interface through which Apex and the Scenario Manager execute commands to the rover via the MSF framework. Viz, the 3-D visualization tool, and ROAMS (dynamic rover model, more details in section 4.3) are tools that were already integrated with MSF.

In one of the scenarios the rover had the task to take contact measurement at three science targets and to take a panoramic picture. Unexpected battery loss was one of the anomalies that were injected during the simulation. The programmed response of the rover was to abort all scheduled tasks other than direct communication to Earth. An expert observer using the REEF software noticed that aborting the measurement with the arm extended automatically invokes an arm stow task. As a consequence, the observer changed the requirements to forbid the arm stow to prevent any tasks requiring power.

The team developing REEF attested to the effectiveness of the MSF since they were able to integrate the robot simulation environment in a matter of days. Because Apex is written in Lisp, they created a Rover Proxy on the Lisp side, which made foreign function calls to objects of the C++ MSF library. Although MSF is capable of running simulations distributed on several computers, the team was able to demonstrate REEF running on a single laptop. Fig. 5 shows the 3D display during this experiment (other REEF components not shown).

4.3. SCIP

While developing MSF, our group closely collaborated with researchers in the Autonomous Systems and Robotics (ASR) at NASA Ames Research Center. During the last couple of years, a major effort has been the development of Single Cycle Instrument Placement (SCIP) capabilities [8]. The demonstration was executed using Ames's six wheeled rover with rocker-bogey suspension named K9 (see model in Fig. 3). The Mission Simulation Facility (MSF) provided the developers of the Planner and the Conditional Executive with a simulation environment to test their plans and software algorithms. Fig. 6 shows the components that were involved in the simulations, and a description of the components specific to this application is given below:

The robot simulation was provided through a high fidelity dynamics model, ROAMS [9], developed at the NASA JPL. ROAMS can compute the kinematics and/or dynamics of planetary type rovers driving on terrains. In addition, the ROAMS library includes several rover software modules that are used in real missions. For this application, ROAMS provided a model of a K9 rover with obstacle avoidance. As with most MSF simulators, Viz represented a view of the rover interacting with the terrain and the science targets. However in this scenario, Viz was also used to define high level plans by selecting targets in the environment. In addition, a camera model simulated the time to take the

Fig. 5. Rover driving on a synthetic terrain while showing its detector range during the APEX experiment.

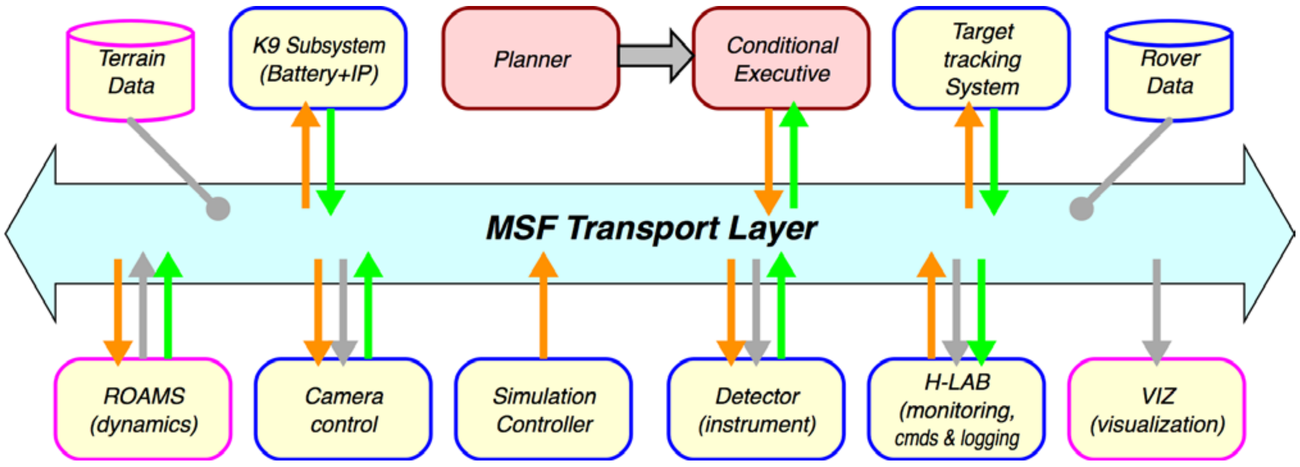

Fig. 6. Simulator configuration to support the SCIP application
(note that the MSF Transport Layer only carries the references to the databases content, not the actual database data).

picture and created a view cone displayed in Viz when a picture was taken.

A model representing the target tracking capabilities was developed as an MSF model. Target tracking was represented as a function of the distance between the rover and the target and the rover's view of the target relative to its stored image. The Conditional Executive switched between branches in response to the current confidence in the tracking system. The "Blue Rock Detector" model was developed to simulate serendipitous detection of special rock formation (e.g. layered). This model "detected" rocks, which exhibited certain attributes and utilities defined in a file, and that fell into the detectors range. The Conditional Executive triggered floating branches based on the status of the detector.

The K9 subsystems that were modeled included power resources and a statistical representation of instrument placement failure. Instrument placement is a complex series of processes that involve navigating the rover to its optimal position, taking an image of the target, finding possible target areas for instrument placement, and finally deploying the arm to place the instrument. Failure may occur during any of these steps. The instrument placement failure statistical model could also be overridden by manually selecting the outcome of the instrument placement.

The MSF simulator was used during several phases of the project. First the advanced features of the Conditional Executive were tested. Then the consistency of plans generated by the planner was evaluated. And finally, during the real demonstration, actual plans were validated in simulation before being sent to the K9 rover.

5. RESULTS AND LESSONS LEARNED

The experiments conducted with MSF based simulators generated two types of benefits for the autonomous software developers: evaluation of the algorithms' behaviors and verification of the software.

First, MSF is used as a platform enabling the rapid test of autonomous behaviors without requiring the hardware platform to be available. This usage ranges from simple runs testing that specific algorithms respond correctly to the situation, to simulation of a full system. For example, during an autonomy technology demonstration at Ames, MSF was used during the field test to validate plans generated by the planner. A precise model of the terrain for the simulation was obtained by using data collected by the actual rover from the Marscape (Ames outdoor Mars analog). This ensured that the simulated rover would encounter the same situations as the actual rover on the real terrain. Before sending a plan to the rover, it was executed in simulation by the same Conditional Executive that ran on-board the rover. The simulation's main advantage is the possibility of testing various branches of the plan by modifying some variables (like available power) or changing the result of some actions (like failure during the arm placement).

Second, MSF was used to test the correctness of the autonomous software (basically discover bugs in the code). MSF simulations of course do not replace formal software validation or testing, but have shown to help discovering problems that often only appear in real situations. For example, a race condition in the Conditional Executive – that had never been seen before on the rover because of the limited number of runs one can actually perform on the real rover– was encountered in simulation. By putting the autonomous software into a number of simulation configurations, it was possible to identify the problem correctly. Another problem at a higher level was discovered in the planner in a similar way. Observing the simulated rover executing a plan with complex branches pointed to some inconsistencies in the plan that led to the identification of a bug (in the plan generation process) that was quickly fixed.

Over four years of simulator development for autonomous rovers, in direct collaboration with users, allowed the MSF team to collect some useful lessons summarized below:

- Simulators at the mission level are definitely missing and researchers in autonomy are eager to use them.
- Simulators supporting research (vs. engineering) require more flexibility to adapt to moving design targets.
- Good analysis of required level of fidelity of the models is critical to avoid wasting time on developing unnecessary high quality components.
- Users most often requested a GUI for setting failure/status on models (tracking, IP, Power model, rover placement) instead of probabilistic models
- The MSF dependency on initially freely available implementation of HLA from DMSO, led to additional work when this middle-ware later became only commercially available.
- Adopting a standard like HLA is good practice, but not necessary if interoperability with multiple simulators is not required.
- Multi-platform support is not enough, multi-language bindings are required. Initially all MSF users were using C++, but recently new projects considering to use MSF are also using Java and C#.

6. FUTURE DIRECTIONS

6.1. Mission Simulation Toolkit

The Mission Simulation Toolkit (MST) is the software package that the MSF has released to the Ames Open Source repository in Spring 2005 [10]. The toolkit includes the HLA-based simulation framework, a library of Communication Objects for the domain of terrestrial surface robot missions, and a number of simulation components. In addition to the components described in Section 3.3 (simulation controller, dynamic engine, locomotor, power model, data analysis/logger), the MST also includes the two other components: 1) an in-house <u>C-Executive</u> that allows scripting simple scenarios in a C like language, and 2) a generic <u>Range Sensor</u> which provides point, line or 2D (depth map) distance information from the sensor to the terrain.

These components provide a MSF user with the basic tools necessary to simulate science mission scenarios involving rovers. Any of these basic modules can be exchanged with higher fidelity models that conform to the same interface or additional models can be connected to extend the simulation capability. For the current release of MST, users need to have access to a HLA Runtime Interface (RTI) implementation. Work is in progress to remove this dependency by integrating MST with the Federated Simulations Development Kit (FDK) [11], freely available from the PADS research group at Georgia Tech.

6.2. End-To-End Mission Modeling and Simulation Environment

End-To-End Mission Modeling and Simulation Environment (EMMSE) is a new project at the Ames

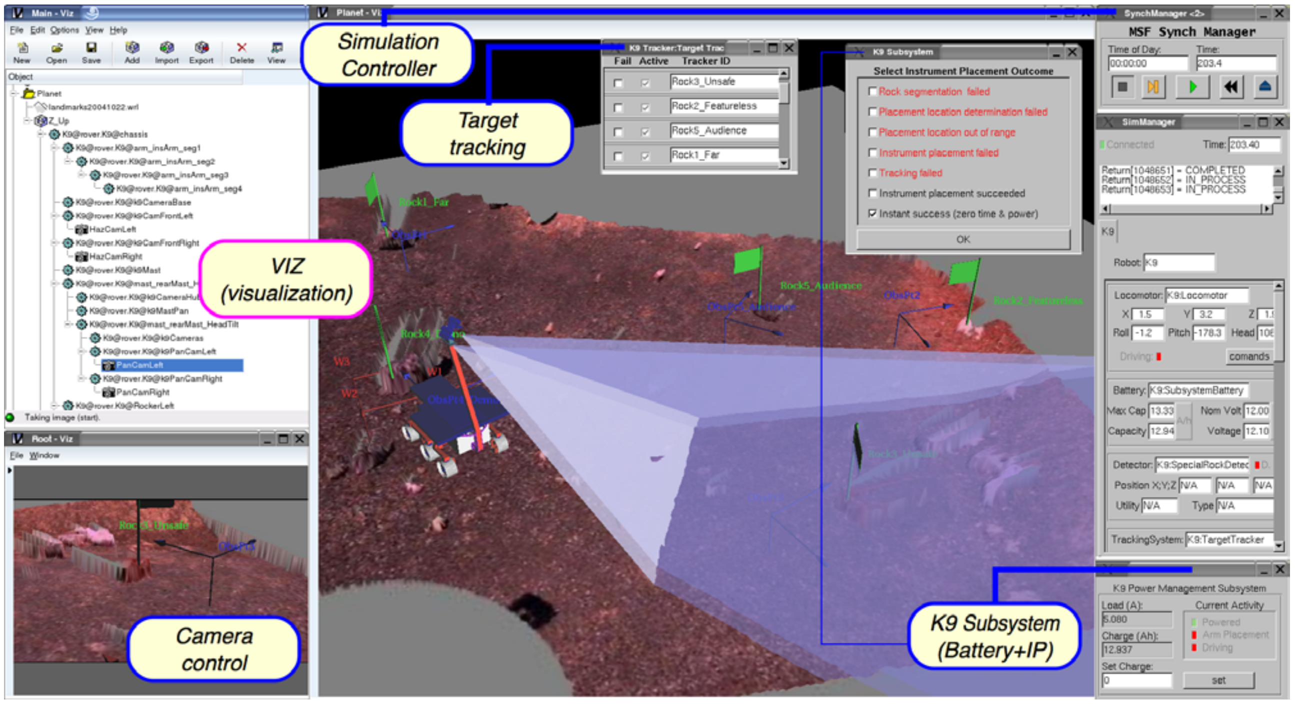
Simulation Controller
Target tracking
VIZ (visualization)
Camera control
K9 Subsystem (Battery+IP)
MSF Synch Manager
SimManager
K9 Tracker:Target Trac
K9 Subsystem
Select Instrument Placement Outcome
Rock segmentation failed
Placement location determination failed
Placement location out of range
Instrument placement failed
Tracking failed
Instrument placement succeeded
Instant success (zero time & power)
OK
K9 Power Management Subsystem

Fig. 7. Displays of several components during a simulation showing a rover taking an image while executing an actual plan. The virtual rover drives on a terrain model generated from data collected by the real rover during the SCIP field experiment.

Research Center that is focused on developing a mixed fidelity end-to-end mission simulation capability to baseline, verify and validate human and robot mission operations. MST will also serve as the simulation framework for integrating a number of new and existing simulators into an environment that we call Mission Operations Design and Analysis Tool (MODAT). MODAT will support modeling and simulation of end-to-end human-robot missions including ground and surface operations (work process and procedures), mission systems, vehicles, robots, space telecommunications and telemetry data.

While analyzing the requirements of the system and of the individual components, several challenges have been identified including support for simulators written in different languages, multiple platforms, event and discrete time-step based time synchronization, mixing high and low fidelity models, and running simulators at different granularity. One of the more potent challenges will be to identify a time synchronization scheme that will support all of the following features: 1) rapid high-level simulation of a days worth of mission activity in minutes; 2) simulation of rover behavior in approximately real time speed for viewing; and 3) simulation of complex processes (e.g. a Moessbauer instrument interacting with an environment feature to create a spectrum) that have high computational demands and that are much slower than the actual measurements. These and other challenges that will undoubtedly be uncovered as the project progresses will provide a critical test for MST. Judging from our experience with tackling other difficult challenges, we are confident that our software will prove itself against these demands and grow while overcoming these obstacles.

7. CONCLUSION

We have evaluated the requirements that are specific to simulations of terrestrial rover missions. We pointed out that low-fidelity models of robotic systems are often sufficient to test autonomy algorithms that make decisions at a high level. We showed that there is a need for being able to replace low-fidelity models with high fidelity models to allow modeling certain processes in more detail. MSF has been presented as a flexible simulation environment that fosters to the needs of researchers who want to test their autonomy software in a mission simulation. The description of the applications of MSF to three specific research problems, each with unique requirements, demonstrates the ability of MSF to adapt to a variety of different systems and demands. We hope in this paper we have demonstrated the convenience, flexibility, and ease with which complex simulations can be created using MSF and how it facilitates integrating existing simulators into a mission simulation.

8. REFERENCES

1. Flückiger, L. and C. Neukom. *A new simulation framework for autonomy in robotic missions*. in *IEEE International Conference on Intelligent Robots and Systems (IROS 2002)*. 2002. Lausanne, Switzerland.
2. Pisanich, G., et al. *Mission simulation facility: Simulation support for autonomy development*. in *42nd AIAA Aerospace Sciences Meeting and Exhibit 2004*. 2004. Reno, NV, United States.
3. Kühl, F., R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. 2000: Prentice Hall.
4. Nguyen, L.A., et al., *Virtual reality interfaces for visualization and control of remote vehicles.* Autonomous Robots, 2001. **11**: p. 59--68.
5. Edwards, L., et al. *VIPER: Virtual Intelligent Planetary Exploration Rover*. in *Proceedings of the 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space (i-SAIRAS 2001)*. 2001. Quebec (Canada).
6. Washington, R., et al. *Autonomous rovers for Mars exploration*. in *Proceedings of The 1999 IEEE Aerospace Conference*. 1999. Aspen, CO.
7. Freed, M.S., M. Remington, R., *Using Simulation to Evaluate Designs: The APEX Approach*, in *Engineering for Human-Computer Interaction*, S.D. Chatty, P., Editor. 1998, Kluwer Academic. chapter 12.
8. Pedersen, L., et al. *Mission Planning and Target Tracking for Autonomous Instrument Placement*. in *IEEE Aerospace 2005*. 2005. Big Sky, Montana, USA.
9. Yen, J., A. Jain, and J. Balaram. *ROAMS: Rover Analysis, Modeling and Simulation*. in *Proceedings of i-SAIRAS 1999*. 1999.
10. NASA Ames Open Source Software, *Mission Simulation Tookit (MST)*, http://ic.arc.nasa.gov/msf/.
11. Fujimoto, R.M., et al. *Design of High Performance RTI Software*. in *Proceedings of the Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications*. 2000.

ACKNOWLEDGEMENTS

None of this work would have been possible without the support of Dr. Butler Hine and the IS program. We would also like to acknowledge the collaborative contributions of several researchers: Dr. Abhinandan Jain and his staff at the DARTS lab, Jet Propulsion Laboratory; Dr. Michael Freed and his APEX team at NASA Ames; and Dr. Rich Washington, Dr. Dave Smith, Dr. Larry Edwards, Dr. Emmanuel Benazera working in the Autonomous Systems and Robotics group at NASA Ames.