

Developing an Autonomy Infusion Infrastructure for Robotic Exploration^{1,2}

Maria G. Bualat, Clayton G. Kunz³, Anne R. Wright³
NASA Ames Research Center
MS269-3
Moffett Field, CA 94035-1000
650-604-4250

Maria.G.Bualat@nasa.gov, [ckunz | awright]@arc.nasa.gov

Issa A.D. Nesnas
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109
818-354-9709

Issa.Nesnas@jpl.nasa.gov

Abstract—Future robotic exploration missions will require autonomy in order to accomplish mission goals for operational efficiency and science return. For example, it will require three communication cycles for the Mars Exploration Rovers, Spirit and Opportunity, to place an instrument on a science target. Reducing this time necessitates highly accurate navigation, obstacle avoidance, target tracking, target analysis, manipulation, and fault diagnosis. Technologies to address these and other operational elements are currently being developed at NASA and within academia. However, infusion into missions has always been a difficult task for researchers. In order to minimize risk, mission managers are reluctant to include new technologies unless they have undergone extensive testing and verification under flight-realistic conditions. Furthermore, infusion of new technologies into missions is made more difficult by the variety of software frameworks under which these technologies are developed. Mission managers would like to see competing solutions demonstrated on a common platform so that they can compare performance and choose the solution best suited to their application.

To address these issues of autonomy infusion, NASA's Mars Technology Program has established an infrastructure for developing, demonstrating and verifying autonomy subsystems to be considered by mission managers for upcoming flights. This infrastructure includes the CLARATy (Coupled Layer Architecture for Robotic Autonomy) [5] software architecture and four rover prototypes: the FIDO, Rocky 7 and Rocky 8 rovers at the Jet Propulsion Laboratory (JPL) and the K9 rover at NASA Ames Research Center (ARC). By providing a common framework for development and mission relevant testbeds on which to demonstrate, the program hopes to better enable incorporation of new technologies to make more capable robotic exploration systems.

A distributed team of engineers and scientists at JPL, ARC, and several academic institutions are developing the CLARATy architecture. As in the open source paradigm,

the goal is to have a large enough developer and user base to foster greater usability. However, having a large, distributed development team also leads to many challenges, especially in the context of a government-sponsored project. In this paper, we will present an overview of the CLARATy architecture and describe the growth of capabilities and algorithms now available within this framework. We will discuss the challenges of developing a software system with remote institutions and the lessons learned in our experience developing CLARATy with ARC, JPL, and Carnegie Mellon University. We will describe the rover testbeds, in particular the K9 rover, and the integration and demonstration of new technologies enabling robust execution, single communication cycle instrument placement, fault diagnosis, and autonomous science.

TABLE OF CONTENTS

.....	
1. INTRODUCTION.....	1
2. THE CLARATY ARCHITECTURE.....	3
3. TESTBEDS – THE K9 ROVER.....	4
4. DEVELOPMENT ENVIRONMENT AND TOOLS.....	6
5. CHALLENGES AND LESSONS LEARNED.....	7
6. K9 INTEGRATION.....	10
7. SUMMARY.....	10
ACKNOWLEDGEMENTS.....	11
REFERENCES.....	11
BIOGRAPHIES.....	12

1. INTRODUCTION

NASA's Space Science Enterprise has developed a roadmap that provides a scientific framework for robotic solar system exploration in the next decade. This roadmap includes ambitious plans to explore the Moon, Mars, and the other planets with a variety of mission sizes and types including: fly-bys, orbiters, stationary landers, rovers, aerial vehicles, and sample and return. Many missions will perform

¹ 0-7803-8155-6/04/\$17.00 © 2004 IEEE

² IEEEAC paper #1327, Version 4, Updated February 3, 2004

³ Contractor with QSS Group, Inc.

intensive in situ exploration of planetary bodies and, in some cases, return samples to Earth for detailed analysis [2]. As missions become more complex and travel farther from Earth, science return and mission safety will require increased autonomy and adaptability through advanced architectures in spacecraft systems [1].

NASA funds robotics and autonomy research at many institutions in the government, private, and academic sectors. Until now, there has been no clear path for these technologies to make their way into missions. Researchers at different institutions use a variety of software frameworks to develop their technologies, making it difficult to compare competing algorithms. A duplication of effort occurs, as each researcher must create a robotic infrastructure in which to develop and test his technology. Additionally, without validation in previous flight systems or flight-relevant field deployments, inclusion of new technologies increases mission risk unacceptably.

In an attempt to address these issues, the Mars Technology Program (MTP) has established an infrastructure for developing, demonstrating and verifying autonomy subsystems for planetary surface robotics. Key elements of this infrastructure are the CLARAty architecture and the

Mars rover prototypes at Ames and JPL.

An example of technology flow into a mission, the Mars Science Laboratory (MSL), is depicted in Figure 1. An MTP led review committee determines which legacy robotics technologies and new technologies coming out of other NASA programs should be ported into CLARAty. The developers of all new technology funded under MTP must deliver their software to the program already integrated within the CLARAty system. The MSL project management reviews the technologies available under CLARAty and sends those it deems most critical to a series of Technology Validation tasks. After validation, MSL project management reviews each technology once more for integration into MDS, the flight software architecture [7].

The development of the CLARAty architecture is an ambitious and ongoing project. It involves a large development team that is spread across the country. CLARAty is designed to include all aspects of rover control, from motor control to planning and scheduling, an extremely large and diverse range of technologies. CLARAty is still a rather young project, and is still very much in a fast-growing stage, with the associated aches and pains that implies.

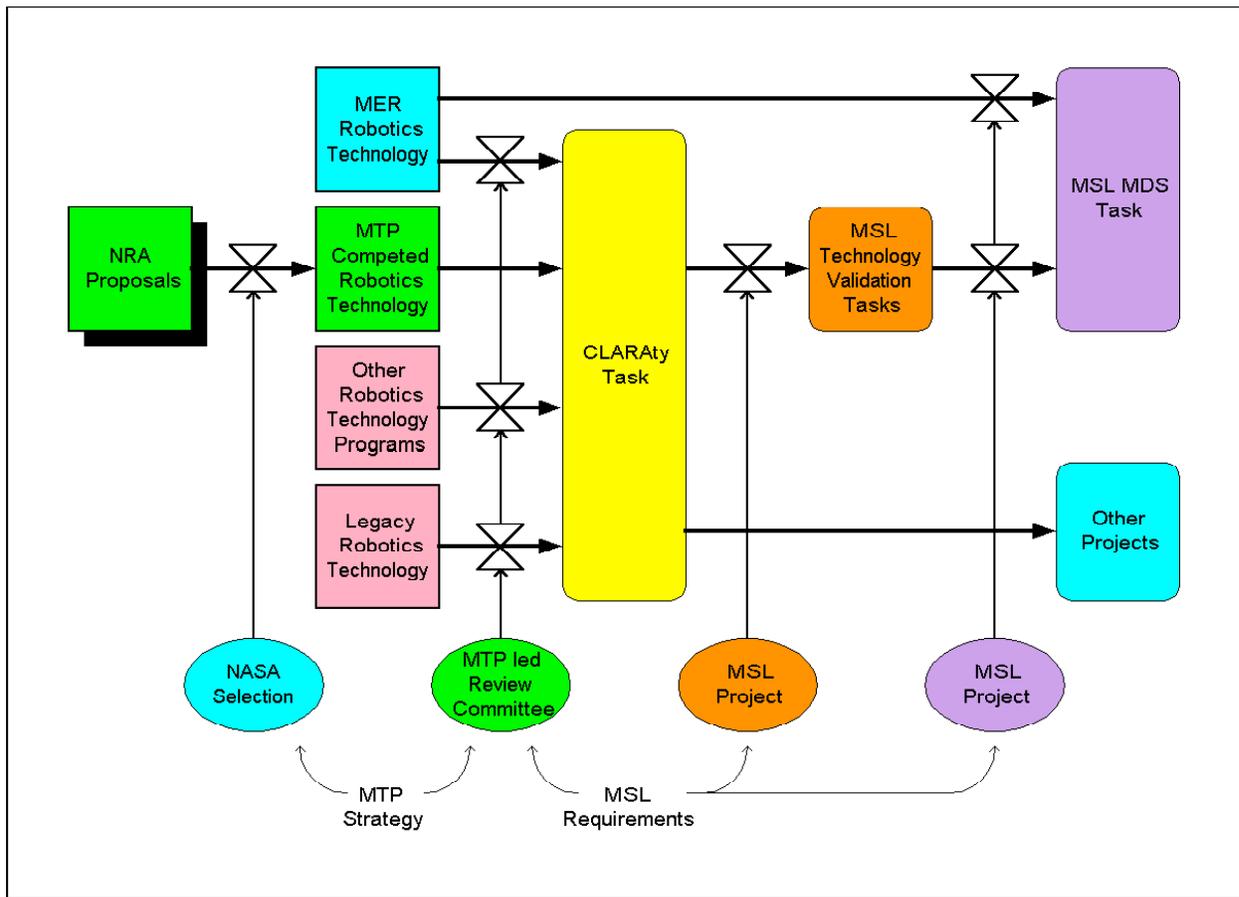


Figure 1 – Rover Functional Autonomy Technology Flow [7]

The K9 rover engineering team at Ames Research Center has been developing and using CLARATy since 2001. In this paper, we discuss our experiences with CLARATy and what we see as the biggest challenges facing the project. In section 2, we will give a brief overview of the CLARATy architecture. In section 3, we will describe the K9 rover testbed and compare it to the other Mars exploration rover prototypes. We discuss the development environment and tools in section 4. Section 5 covers some of the challenges and lessons we've learned in our experience with CLARATy, and in section 6, we give some examples of technology we've integrated and demonstrated onboard the K9 rover, especially concentrating on the experience of integrating software into the CLARATy environment and adapting it to a new platform. Finally, we summarize in section 7.

2. THE CLARATY ARCHITECTURE

CLARATy is a unified and reusable software architecture that provides robotic functionality and simplifies the integration of new technologies on robotic platforms. It is a research tool designed for the development, validation, and maturation of various research technologies [5].

One of our goals is to provide a design that allows researchers to use various components spanning domains outside their immediate expertise. These components should be flexible and extendible to support various applications. We need to use well-understood and well-developed knowledge from the various domains and adapt them as generalized and reusable components. Just as an operating system provides a level of abstraction from the computational hardware, our goal is to provide a level of abstraction from the robotic hardware implementation that will allow developers to "integrate once and run anywhere." Of course, there are physical limitations to this goal that result from the large variability in rover capabilities.

CLARATy is a domain-specific robotic architecture designed with four main objectives: (1) to reduce the need to develop custom robotic infrastructure for every research effort, (2) to simplify the integration of new technologies onto existing systems, (3) to tightly couple declarative and procedural-based algorithms, and (4) to operate a number of heterogeneous rovers with different physical capabilities and hardware architectures.

CLARATy is a collaborative effort among many institutions including the Jet Propulsion Laboratory, Ames Research Center, Carnegie Mellon University, and the University of Minnesota. Through NASA's Mars Technology and Intelligent Systems programs, these and other universities, centers, and members of the robotics community are contributing and integrating algorithms into CLARATy. CLARATy builds upon decades of robotic experience at the various robotic centers.

It is our goal to have the resultant architecture and software algorithms made available to the larger robotics community through an open source distribution.

Background

Developing intelligent capabilities for robotic systems requires the integration of various technologies from different disciplines. Robotic control systems require the interaction of various software components within a real-time system, and the management of uncertainties resulting from the interaction of the robot with its environment. Environmental uncertainties, the complexities of software/hardware interactions, and the variability of the robotic hardware make the task of developing robotic software complex, hard, and costly. Hence, it has become increasingly important to leverage robotic developments across projects and platforms. Because many algorithms developed for robotic systems can be generalized, it is possible to use these algorithms on various platforms irrespective of the details of their implementations.

With the increased interest in developing rovers for future Mars exploration missions, a significant number of rover platforms have been designed and built over the past decade. Several NASA centers and university partners use these platforms to test their newly developed technologies in order to improve the autonomous robot capabilities. Because of isolated software development efforts, exacerbated by differences in the mechanical and electrical designs of these vehicles, they have historically shared little in terms of software infrastructure. As a result, transferring capabilities from one rover to another has been a major and costly endeavor. Furthermore, because robotics systems cover several domain areas, researchers of a single domain also needed to integrate their newly developed technology into the complex robotic environment. Proper integration requires an in-depth understanding and characterization of the behavior of various components of the system, which may vary from one platform to another.

Description

The CLARATy architecture has two distinct layers: the Functional Layer and the Decision Layer (Figure 2). The Functional Layer uses an object-oriented system decomposition and employs a number of known design patterns to achieve reusable and extendible components. These components define an interface and provide basic system functionality that can be adapted to a variety of real or simulated robots. The Functional Layer includes a number of generic frameworks centered on various robotic-related disciplines. Packages included in the Functional Layer are: digital and analog I/O, motion control and coordination, locomotion, manipulation, vision, navigation, mapping, terrain evaluation, path planning, science analysis, state estimation, simulation, and system behavior. The Functional Layer provides the system's low- and mid-level autonomy capabilities. Control algorithms such as vision-based navigation, sensor-based manipulation, and visual target tracking that use a predefined sequence of operations are often implemented in the Functional Layer. In some cases though, it is possible to generate such sequence of operations by modeling them as activities and having the Decision Layer schedule instantiations of these activities based on appropriate mission goals and constraints.

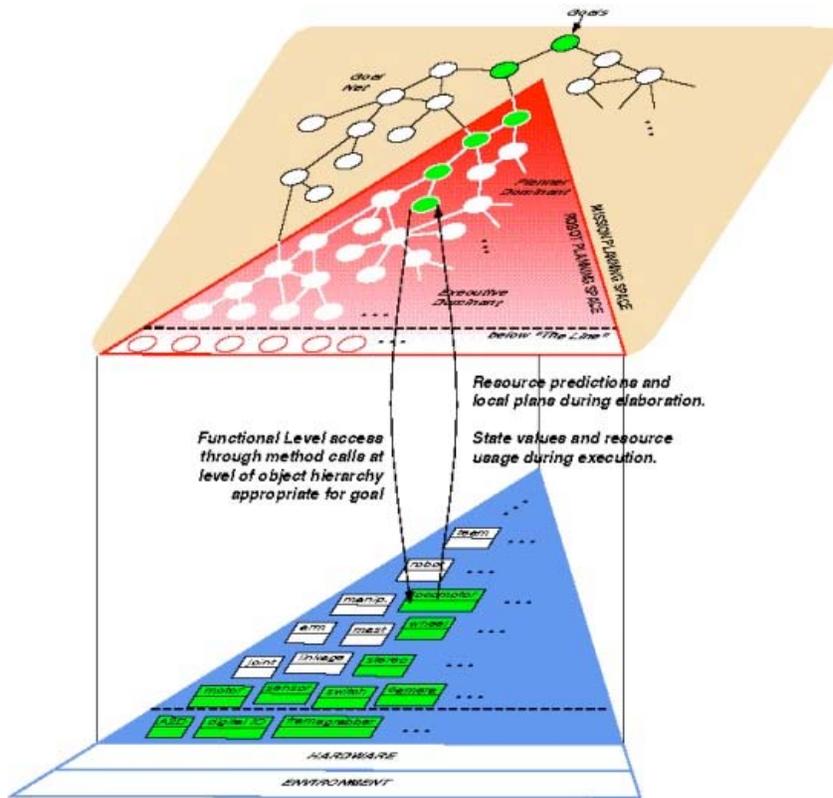


Figure 2 – The CLARAty architecture with top Decision Layer and bottom Functional Layer.

The Decision Layer is a global engine that reasons about system resources and mission constraints. It includes general planners, executives, schedulers, activity databases, and rover and planner specific heuristics. The Decision Layer plans, schedules, and executes activity plans. It also monitors the execution modifying the sequence of activities dynamically when necessary. The goal of a generic Decision Layer is to have a unified representation of activities and interfaces.

The Decision Layer interacts with the Functional Layer using a client-server model. The Decision Layer queries the Functional Layer about availability of system resources in order to predict the resource usage of a given operation. The Decision Layer sends commands to the Functional Layer at various levels of granularity. The Decision Layer can utilize encapsulated Functional Layer capabilities with relatively high-level commands, or access lower-level capabilities and combine them in ways not provided by the Functional Layer. The former is valuable when planning capabilities are limited, or when under-constrained system operation is acceptable. The latter is valuable if detailed, globally optimized planning is possible, or if resource margins are small. CLARAty supports both modes of operation. Status on resources, state conditions, and activity execution are reported from the Functional Layer to the Decision Layer asynchronously or synchronously at rates specified by the Decision Layer.

3. TESTBEDS – THE K9 ROVER

Currently, rovers at three institutions are running under the CLARAty architecture: K9 at Ames Research Center (Figure 3), FIDO, Rocky 7, and Rocky 8 at the Jet Propulsion Laboratory, and Bullwinkle, a commercially-available ATRV robot [12], at Carnegie Mellon University. JPL also has several benchtop testbeds that simulate the computational and some motor subsystems of the various rovers. Operating systems actively supported under CLARAty include: VxWorks, Sun Solaris, and RedHat Linux. Parts of CLARAty have also been built under Mac OS X, Windows NT/2000, and IRIX.

K9, FIDO and Rocky 8 are currently the highest fidelity Mars rover prototypes for upcoming NASA missions available, and are based on the same 6-wheel steer, 6-wheel drive rocker-bogey chassis designed by JPL. They are all outfitted with electronics and instruments appropriate for supporting research relevant to remote science exploration. Each rover has different avionics and instrumentation, with some overlaps between the rovers (for instance, K9 and Rocky 8 use the same IEEE 1394 cameras for navigation and obstacle avoidance).

The computing onboard K9 is performed by a 1.4 GHz Pentium-M laptop running the Linux operating system. The avionics of the K9 rover have been specifically designed to enable simulation of realistic mission power constraints and science operations. Low power subsystems that can be commanded on and off and Li-ion batteries with state of

charge monitors allow planning and execution systems on-board to reason and act on resource availability. An auxiliary microprocessor communicates with the main computer over a serial port and controls power switching and other I/O processing. The rover has a 5 degree-of-freedom arm with a reach of 70 cm that is based on the FIDO rover arm design.



Figure 3 – The K9 rover places an instrument against a rock in the NASA Ames Marscape.

Instruments on-board K9 include a compass, an inertial measurement unit, and 3 pairs of monochromatic cameras used for navigation and instrument placement. A differential GPS unit provides ground truth for navigation systems. A scanning laser rangefinder with coverage of the arm workspace is also available to provide extremely high-resolution 3D models for instrument placement. Current science instruments include a pair of high-resolution color cameras mounted on a pan/tilt unit atop a fixed mast approximately 1.5 meters above the ground, and the CHAMP, an arm-mounted focusable microscopic camera developed by the University of Colorado, Boulder [4].

One of the main challenges in developing generic components and adapting them to different robots stems from the variability of the platforms and their capabilities. We will use the example of wheeled locomotion to illustrate how to use domain knowledge to classify vehicles to enable the development of generic and reusable classes. We will also discuss the challenges that arise from adapting the generic algorithms to a number of rover platforms with different hardware architectures.

Wheeled locomotors have different capabilities depending on their mechanical configuration. Consider the locomotion capabilities of a number of mobile platforms, the ATRV, Rocky 7, Rocky 8, FIDO, K9, and Sojourner rovers (Figure 4). These wheeled vehicles have different maneuvering capabilities. The proper classification of these vehicles is based on the domain knowledge of the kinematics and dynamics for controlling these vehicles. One can classify these as non-steerable (or skid steerable) such as the ATRVs, partially steerable such as the Rocky 7 and Sojourner rovers, and fully steerable such as the Rocky 8,

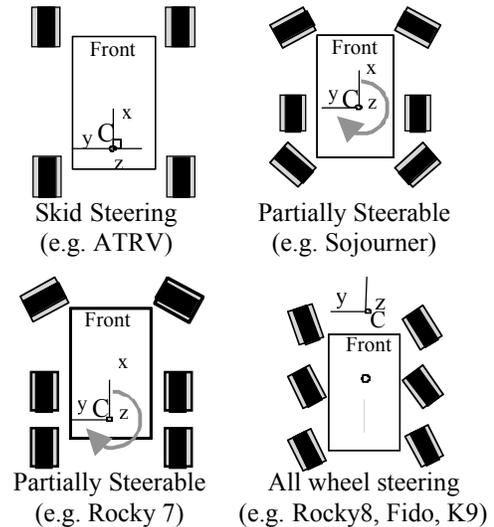


Figure 4 – Rover Locomotion Types

FIDO, and K9 rovers. Partially steerable vehicles can have different configurations. For example the Sojourner rover has six drive wheels and two non-steerable center wheels. On the other hand, Rocky 7 has only two steerable front wheels. As such, partially steerable, wheeled locomotors are constrained to instantaneously move about a rotation center that lies along the non-steerable wheel axle (or a virtual axle that averages all non-steerable axles in order to minimize slip). Fully steerable vehicles can perform crab maneuvers, controlling trajectory and vehicle orientation independently. Partially steerable vehicles have more constraints and cannot independently control path and heading, but can use a “parallel parking” maneuver to achieve a crab equivalent [10].

A second challenge that arises in addressing these classes of vehicles comes from the accessibility to the system’s control parameters. For example, the ATRV provides independent control for each side of the vehicle only but not for each individual wheel. The control model for this vehicle is different from those vehicles where each wheel can be controlled individually.

A third challenge stems from the different motion control architectures. Consider the motion control architecture of Rocky 7, Rocky 8, K9 and FIDO (all have six wheels and almost all have full steering capabilities). While closer in resemblance to each other than to the ATRVs, for instance, the control architecture for each vehicle is still unique. Both the Rocky 8 and K9 rovers (Figure 5) use a distributed motion control architecture where each motor interfaces to a single-axis microprocessor controlling the motor servo loop and, in some cases, profiling a trajectory. Distributed microcontrollers can, as in the case of Rocky 8, also perform analog and digital I/O operations. They also possess some additional programmable processing capabilities. In a distributed system, microcontrollers are connected to the main processor via some type of a serial bus. The K9 rover uses a multi-drop RS-422 serial link for the control of its mobility motors. Rocky 8 uses a single I2C bus for its locomotor, arm, and mast subsystems. There

is an important coupling between the arm/mast and the locomotor as a result of the shared bus. The software architecture has to enable the simultaneous operation of the manipulator and locomotor subsystems by managing the shared resource. While in an abstract sense the two subsystems are independent, the implementation is not.

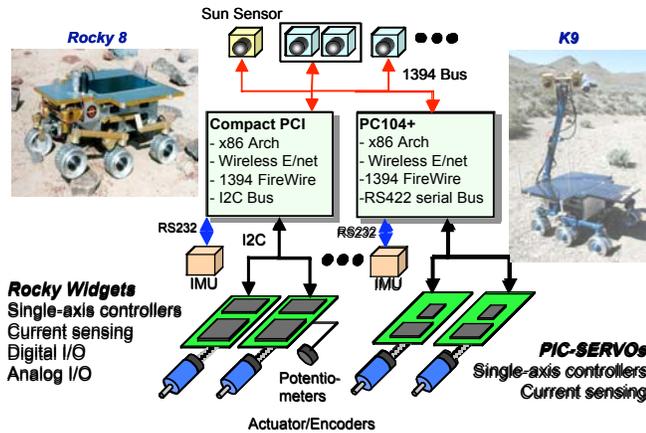


Figure 5 – Distributed motion control architecture for Rocky 8 and K9.

Another aspect of hardware architecture is hardware synchronization. The K9 system supports hardware synchronization of motors via broadcasting serial commands that tell all axes to synchronously execute their loaded trajectory, or synchronously stop. The Rocky 8 rover implements synchronization in software by loading all motor trajectories first and then issuing start commands to all motors sequentially to minimize latency between the first and last motor. Once again the software architecture should support these two different modes of synchronization. As such, support for device groups is an essential part of the CLARAty architecture. The flexibility in the implementation of group commands is also important since hardware implementations can vary dramatically.

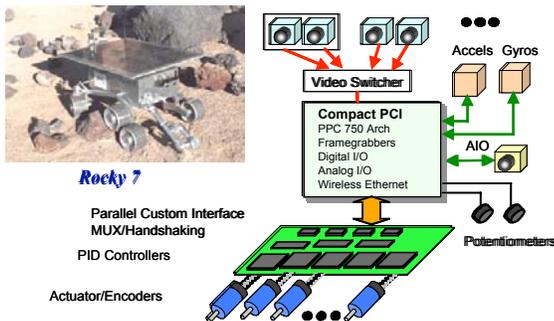


Figure 6 – Custom parallel bus for the multiplexed motion controllers on Rocky 7.

The Rocky 7 system uses commercial-of-the-shelf (COTS) microcontroller chips (LM629) for the motor control (Figure 6). These controllers are laid out on a central motion control board and are connected to the host processor via a custom parallel port connection with chip multiplexing. All

actuators in the system share the same bus, but the communication bandwidth is higher than the serial links for both Rocky 8 and K9. Similar to Rocky 8, this motion control board supports the locomotion and manipulation subsystems. As in the case of Rocky 8 and K9, the closed loop servo control is done on the microcontrollers that have fixed control law with programmable parameters and modes.

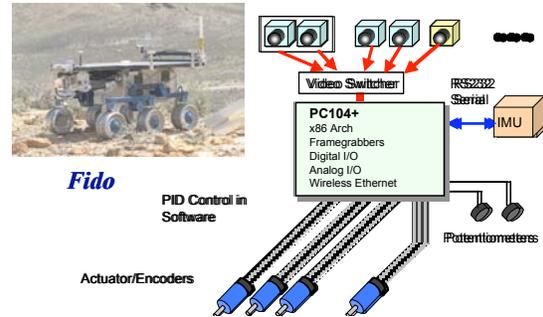


Figure 7 – Centralized memory-mapped motion control architecture for FIDO.

Figure 7 shows a third implementation of a motion control architecture. The FIDO rover [11] uses a centralized hardware-mapped control architecture. The motors are directly connected to an analog output board and the encoders are directly connected to a quadrature encoder board. All hardware states and registers from the PC104+ boards are mapped via the PCI backplane to the host processor's memory making them readily accessible to the software. There is virtually zero cost from a software architecture standpoint to retrieve the value of any register as compared to the other systems. Hence the coupling among the various motor/encoder states is abstracted by the hardware. However, since there is only one processor (host) in the system, the servo loops for all actuators have to be done on the main processor. This introduces a coupling between the servo control of the motors and the application algorithms that will be competing for the same computational resources. It also places a requirement on the operating system and the software architecture to meet hard real-time scheduling guarantees. So while the K9 and Rocky 8 rovers can operate in a soft real-time environment such as Linux, the FIDO rover requires the operating system and supporting architecture to run in hard real-time. On the other hand, the FIDO architecture has the advantage of allowing the software to easily modify the control law and insert validation checks in case a motor or encoder failure occurs.

4. DEVELOPMENT ENVIRONMENT AND TOOLS

The CLARAty software development process is distributed among three primary institutions: Jet Propulsion Laboratory, NASA Ames Research Center, and Carnegie Mellon with active participation of various universities including University of Minnesota, University of Washington, MIT, and University of Michigan. To support the various institutions and teams, CLARAty supports a variety of host and target platforms. The host platform is the type of computer that the user runs the tools on to

create, modify, and build software. CLARATy works on a variety of Unix host platforms, the most common being Linux and Solaris. The target platform controls the robot or simulation, and is most commonly Linux or VxWorks.

To support building for multiple targets, platforms are named by the processor type, OS type and version, and compiler type and version, such as sparc-solaris2.7-gcc3.2 or ix86-vx5.5-gcc2.95. In the case of Linux, a slightly modified algorithm is used since there is not a clear 'version' of Linux as there is with Solaris or VxWorks, and the determinant of binary compatibility depends on the versions of gcc and glibc, not the version of the kernel. For example, a stock RedHat 7.3 installation would be ix86-linux-gcc2.96-glibc2.2, and a stock RedHat 9 installation would be ix86-linux-gcc3.2-glibc2.3.

The development environment and tools break down into two types: host tools, and target packages. Host tools used by CLARATy include:

- YaM (Yet another Make), a module/version/build management tool developed at JPL
- perl [13], required by YaM and used in the build
- CVS (Concurrent Versions System) [14], a version control system used explicitly in modules, implicitly by YaM
- make [15], controls the build in modules and at the top level
- doxygen [16], creates HTML documentation from sources

Target packages used by CLARATy include:

- ACE (Adaptive Communication Environment) [17], an operating system adaptation layer
- cppunit [18], a testing support infrastructure
- Qt [19], a user interface toolkit used only on non-VxWorks targets

The CVS and make tools are universal and stable enough that they can safely be assumed to already be present on the host computer and in the user's path. The other tools and packages for supported hosts and targets are available in a central location in the CLARATy AFS (Andrew File System) [20] space at JPL. Placing these files in AFS allows any computer equipped with an AFS client to access them at a canonical location: `/afs/jpl.nasa.gov/group/claraty/pkgs`.

For each site where CLARATy is used, there is a SOURCEME file that users must source in their shell in order to set up the needed environment variables. This SOURCEME file detects the host platform type, and uses that to set up appropriate executable, shared library, and perl module paths to allow the host tools to be used, as well as base paths to the target packages and various configuration files. Having the tools and packages available in a central location with a uniform path from anywhere is a very good thing in that it means that CLARATy development is not tied to just particular computers, or a particular subnet; it

works from anywhere. However, using the tools and packages from JPL's AFS server from remote locations is slow. For occasional use, this is not a serious problem. However, if there are multiple heavy users, as is the case at Ames, this can become a significant limitation. It is therefore advantageous to mirror the tools and packages locally and setup the site-specific SOURCEME file with local overrides for the base paths for the tools and packages. This significantly increases performance. However, the CVS repository is also located in JPL AFS space, and cannot be mirrored.

Once a user has setup his environment, he can use YaM, CVS, and make to create, maintain, or build a "sandbox" containing a set of CLARATy modules. A sandbox contains a local copy of work modules that a user may modify, and symbolic links to pre-built link modules. The CLARATy makefiles (files that control the actions of make) use the environment variables setup in the SOURCEME file and any additional environment variables set up by the user to determine the target platform, and find the header files and libraries in the target packages to use as part of the build. Different make targets allow the users to either build target executables, doxygen documentation, or perform various administrative tasks. If a user develops changes that he wants to check in or release, he can use a combination of CVS and YaM to do so, though there are complex issues regarding testing, target support, and impact on other modules or users. These issues are discussed in the next section.

5. CHALLENGES AND LESSONS LEARNED

We may categorize most of the challenges we have encountered during CLARATy development as springing from three primary sources: physical distance between team members and testbeds, size and complexity of the software, and governmental regulations. In some cases, solutions have been found or can be foreseen. In others, we are uncertain a workable solution can be achieved.

Distributed Team Challenges

For most of the K9 team, CLARATy was a first-time experience in distributed development of a large software project. The software must be built on several software platforms – VxWorks, Linux, and Solaris – with different compilers, and must work on many different robotic platforms. Perhaps the biggest barrier to rapid development has been the difficulty of verifying the impact of software changes in a given module on other modules, platforms, and robots which the person making those changes either does not normally use, or which are not available in the local lab. The CLARATy project has established a testbed containing hardware subsystems of the rovers that is accessible via remote login. However, some hardware is one-of-a-kind, too expensive to replicate in the testbed, or burdened with restrictive licensing.

It is a significant policy challenge to determine how to deal with this issue, and one which is tied in to the question of how users should structure and update their sandboxes. We

have attempted to pursue a policy of users primarily using pre-built libraries and binaries that are supported by the YaM tool to reduce the amount of software that needs to be compiled for development and testing. These pre-built modules which the users are not actively modifying are known as “link modules” (described below in more detail). However, this leads to the requirement that the developer releasing a module has to build and test it on all supported platforms, which includes VxWorks targets. The net effect of this requirement has been that releasing a module is intimidating, slow, and difficult.

For example, imagine that a developer makes significant changes to the camera Generic Physical Component (GPC) and wants to make a release. This necessitates modification, testing, and re-release of the modules for all of the hardware specializations being used on the various rovers. However, not all the cameras in use are available either locally or in the CLARATy testbed. For instance, the interface board for Rocky 8’s cameras is only available on the rover itself. Testing those cameras requires the interactive support of a person co-located with the equipment, thus doubling up on the personnel required had the device been available locally. If testing on one such camera points out changes required to the GPC, then the process must begin once again.

Even for hardware that is available in the testbed, we have found it difficult to automate testing, particularly tests that require access to physical components, or tests requiring VxWorks, due to a shortage of platforms, requirement for exclusive access (VxWorks is a single user system), and difficulty automating kernel configuration and loading. Finally, the fairly high latency of the network between the AFS servers at JPL and Ames also slows the testing process.

As a result of these issues, developers tend to delay integration into the main tree of the repository, as each site is encouraged to develop pieces of code that are only relevant locally. As the project grows, the reluctance among the developers to commit changes back to the main repository will only increase, as testing across platforms, re-integration, and operation across a slow AFS link will continue to take more and more time.

Weekly or bi-weekly conference calls involving the entire CLARATy team help to keep everyone abreast of status and issues. Teams working on subsystems communicate regularly via phone and email. However, we have found that face-to-face meetings among the team members are an invaluable way to keep work flowing and synchronized. JPL hosted a CLARATy workshop in early 2003 that most of the team was able to attend. While these meetings are expensive given the amount of time and number of people involved, they and smaller ones like them are essential if any sort of team coherence is to be maintained.

Size and Complexity Challenges

Over the past two years, the capabilities and algorithms encompassed by CLARATy have grown at a rapid rate. At

the time of the writing of this paper, there were over 300 modules and packages in the CLARATy database. Managing the database and keeping builds consistent becomes extremely complicated in the face of inter-module dependencies and the use of link modules (described below).

CLARATy is broken up into small modules, most of which depend on sets of other modules. This makes it possible for each user to use only the parts of CLARATy he or she needs, ignoring the rest, and have each part rest in a consistent framework. The module decomposition in CLARATy is based on the reusability model of the architecture. In addition to making the usage of CLARATy more efficient, it is a necessity since CLARATy supports numerous drivers for various types of hardware and rovers not relevant to all users. This modularity scales the complexity of the sandbox according to the application. For example, if Bob wants to do image analysis, and Jane wants to drive motors, each can do so without having to check out and build other parts of the system. Bob will have image, camera_model, etc. checked out in his sandbox, Jane will have motor, device, etc. checked out in hers, and both will have the matrices and share modules checked out, since these modules are used by both. If later they want to put their work together to drive Jane’s motors using the results of Bob’s image analysis to control where to go, the fact that they are both using CLARATy makes this easier than if they had worked in isolation, since they are using the same math types and they can check out the modules together in the same sandbox in a consistent manner. However, managing the inter-module dependencies, choices of versions, and link modules vs. work modules (and their branches) can be quite complex.

Each module release results in both a new release tag for the source code and the creation of a ‘link module’. The person creating the release builds and tests on all the supported platforms, and YaM moves the module directory containing the sources, libraries, and executables, into a canonical release area. When checking out or modifying a sandbox, users have the option to use that module as a link module, meaning that their sandbox contains symbolic links to that pre-built module directory, or a work module, meaning that their sandbox contains a checkout in a local directory of the source files as they were at the time of that particular release. The sources are built in the context of the user’s sandbox. Link modules are faster to use, since the user doesn’t have to rebuild them, but are somewhat risky to use if the version dependencies are incorrect. Incorrect version dependencies can lead to *build skew*, an error caused by linking together objects that use differing versions of header files. Build skew is dangerous and difficult to detect. It may show up as link or dynamic loading errors that can be difficult to track down, but that correctly prevent the user from creating or running a skewed executable. It may also manifest as subtle runtime corruption that may cause seemingly random and untraceable crashes.

In its original form, YaM manages link-module version dependencies by requiring users to release packages in addition to releasing modules. A package is a collection of modules containing all the dependencies. For example, if

module A depends on module B and A is in the package P, B must also be in P. A package release groups together a consistent set of release versions for all the component modules. If A-1 and B-2 are in package release P-1 and A depends on B, then the header files in B-2 must be the same as those that A-1 was built against in order to be consistent.

This model works fine for distinct projects that share some overlap in the modules they use, but that have non-overlapping user bases. However, this is not the case with CLARATy. There are many components, and different people, or even the same person at different times, use them in fluid combinations. The package release model does not scale well for this type of project. In the previous example, Bob and Jane were originally using image-related modules and motor-related modules in isolation, then at some point decided to put them both together into the same sandbox. For this to be possible, it would require three packages: one containing image modules, a second containing motor modules, and a third containing both image and motor modules. However, there are dozens of such related module sets that might be used alone or in combinations, plus several specializations for each GPC (the motor specialization Jane would have checked out depends on what hardware she uses). The number of permutations is very large.

In an attempt to address this problem, we have modified YaM to automatically track the module version dependencies each time a module release is made by recording module versions present in the sandbox when the resulting link module was built. This preserves the information needed to detect potential build skew, and the user can, if sufficiently skilled and motivated, examine the version dependencies and logs and decide whether or not a given sandbox is consistent. However, in practice it is extremely challenging to use this information to analyze sandboxes and to decide how to combine partially overlapping sets of modules together, as in the earlier example. Imagine that Bob is using image-R1-04b as a link module, which was built against arrays-R1-08a. Jane is using motor-R1-06b as a link module, which was built against arrays-R1-08b. A header file in arrays changed between those two versions, so one cannot assume that one can safely use all of those link module versions together. It turns out that there are more recent releases of image (R1-04b through R1-04c-Build01) which were built against the more recent version of arrays, but in which an image header file changed. It is a complicated question as to whether or not Bob can safely upgrade his image module, and if so to which version, since he may be using other modules which may or may not have been re-released using the newer version of the image header files.

Attempts to make tools to automate this process, or to assist the user, have met with difficulty due to the complexity, high branching factor, and difficulty of presenting the options in a way that a user can understand. Ultimately, the right option to pursue depends on subtle considerations of what the user wishes to accomplish, which is difficult to convey to a software tool. For example, another possible way for Bob to merge his work with Jane's would be to check out an older version of motor that used

the same version of arrays (arrays-R1-08a) as his image module. If Jane's work depends on the newer version of motor, or if there isn't a release of the motor specialization she's using that uses the older version of arrays this may not work. Depending on the changes to the array header file, Bob could also use the same releases of motor and the motor specialization as Jane has been using as work modules instead of link modules. These types of considerations can be quite subtle and are outside the scope of what an automated tool can take into account.

These issues illustrate a fundamental complexity of an interdependent modular system in which changes can be made to modules independently. This is not unique to CLARATy but is seen in software projects that support a similar reuse model with numerous adaptations. This problem is magnified by the fact that it is extremely difficult to test the impact of the changes to one module on all the modules that depend on it, due to limited hardware availability, slow build times, and an essentially unbounded number of modules, not all of which it is possible or, in some cases, even desirable to maintain. These considerations are also directly at odds with the desire to reduce the burden on the developer to allow rapid development. We have not yet found a workable solution to this, and believe it is a fundamental problem CLARATy, or in fact any similarly modular software development effort, faces.

Governmental Regulation Challenges

Until mid-2003, CLARATy was unclassified by the U.S. State Department with respect to the ITAR restrictions on technology export. Several key principle investigators (PI) at Ames are foreign nationals, and because of the lack of classification they were unable to use our rover while the project was under review. Any software they needed run on the robot had to be written by a U.S. citizen team member, and the PIs were unable to assist in debugging and testing.

Fortunately, CLARATy has now cleared both the State Department and the Commerce Department with only minimal restrictions on its distribution. This enables many of our collaborators to begin incorporating their technologies onboard the rover.

Other Challenges

Another difficulty faced by the project stems from the fact that each institution tends to focus on the platform/rover that it uses locally. JPL uses VxWorks with FIDO, Rocky 7 and Rocky 8, the AI group at JPL uses Solaris without any particular rover, and the Ames group uses Linux with K9. Because of this division, each center tends to push development on their own platform, without much heed paid to the quirks of the other platforms. For example, until recently the VxWorks platform was restricted to a very old version of the g++ compiler, which didn't support many features that those programming under Linux and a more modern compiler were used to using. Each time a developer at Ames wanted to make a change, he had to verify that the

older compiler supported it, and work around as-yet-unimplemented language features.

In addition to supporting CLARAty, most rovers and their teams are also tasked with supporting other technology development projects. Due to limited resources, this tends to drive the developers to add capabilities to the system as they are needed for a particular problem rather than in a more thought out manner. While added resources could help alleviate this problem, one of the strengths of CLARAty is its responsiveness to the needs of the rover technology community, and as such, the project should continue to involve active robotics researchers in its development team.

6. K9 INTEGRATION

Since we started participating in the CLARAty development effort, many parts of the K9 software base have been integrated into CLARAty. Most encouraging is the fact that many parts of CLARAty that did not originate in the K9 software base are now being used on K9, and many of the K9-specific pieces of code are specializations of abstract CLARAty types and interfaces. At the present time, only the CLARAty functional layer is being used on K9, but as we move into the more abstract realm of rover functionality, the interfaces will become general enough that higher-level decision layer elements will be able to interface with the K9 hardware in the same way that they interface with the other rovers.

K9 - CLARAty integration started at the lowest level, in the realm of motors, cameras, busses, and in the physical and geometrical description of the hardware itself: the physical relationship between the joints and wheels, and the orientation of the various sensors, including camera calibration. This is an ongoing process, as we revise the way to describe a motor or compass or IMU or camera in order to achieve greater functionality or compatibility and attempt to re-integrate it into CLARAty as a standard. Once an initial implementation of the basic robotic building blocks was available, it became possible to describe whole rover systems more generally, for example, motor -> wheel -> locomotor, and similarly, motor -> joint -> manipulator.

As we move up the abstraction tree, not only do more things become possible; they also become easier. With a uniform interface for locomotor, for example, which allows one to generically specify drive commands and sequences, it was relatively easy to start using the CLARAty Morphin navigator developed at CMU [21], which adds obstacle avoidance, and only needs to know about three things, all specified generically: a locomotor, a model describing what the locomotor can do, and a source of 3-D point clouds from the environment. In K9's case the point clouds are provided by a pair of stereo cameras mounted to a pan/tilt unit, but in other cases by fixed cameras pointing in different directions, or by a laser scanner. Similarly, once the interfaces are complete, one will be able to attach generic position sensors (compasses, IMUs, GPS units, odometry) to a pose estimator that can perform localization using any number of techniques.

Visual tracking of features in the environment (typically rocks or navigation landmarks) is one of the more advanced technologies now in CLARAty. In this case, the developers started with a specific set of algorithms to try on the various rovers, and quickly implemented them. Later, in order to make a comparative study, we stepped back to design a generic interface that one could use to communicate with any algorithm that tracks 2-D features in images, and re-shaped the interface of the first tracker to conform to it. At the same time, we took the visual tracker that was previously implemented for K9, and re-shaped its interface to also conform to the new CLARAty generic base class. The result is several tracking algorithms that can be easily swapped and compared, providing easy demonstration of capabilities for mission planners.

CLARAty subsystems used by K9 include:

- cameras and camera bus;
- motors;
- joint, manipulator, wheel;
- locomotor;
- navigator;
- power system, including battery, solar panel, and charger;
- kinematics model;
- linear algebra;
- camera model representations;
- low-level subsystems, such as serial communications.

There has been considerable collaboration and these systems include items developed both at Ames and elsewhere. It has often been the case that the seeds of something required to answer the needs of a technology development task on K9 exist in CLARAty. We would use these, sometimes redesigning the interfaces to make them work for our application, and then try to reintegrate the new interfaces back into CLARAty. It would be preferable to design new modules as a team so that re-integration becomes smoother. As CLARAty matures, the occurrence of such ad hoc integration should lessen.

7. SUMMARY

The experience of integrating CLARAty onto the K9 rover has been one of many challenges and yet is one that the people involved would rate as rewarding. As CLARAty grows to include more algorithms and better tools for distributed development, the payoffs will only increase.

Ames experienced an example of how well the system can work in our integration of the Morphin navigator onto K9. In just two months, a single engineer was able to integrate and test CMU's locomotor and navigator code onto the rover. This software now runs on rovers at three different institutions.

The true payoff of CLARAty for Ames has been the ability to demonstrate new technologies within mission operational

scenarios without having to develop the entire system in-house. For example, leveraging manipulation and navigation capabilities available within CLARAty, we have been able to demonstrate autonomous single-cycle instrument placement [6].

As the CLARAty community grows, mission managers will have an increasing set of capabilities that they can then compare and contrast to find those technologies that best enable their mission. However, the difficulties created by the sheer size and complexity of the system that we have already seen will be exacerbated. The key will be to control growth and find or develop tools that ease testing and version control.

ACKNOWLEDGEMENTS

The authors of this paper would like to thank the researchers and developers of CLARAty at JPL, Ames, and CMU.

The research described in this paper was carried out by the Intelligent Robotics Group at the National Aeronautics and Space Administration's Ames Research Center; the Jet Propulsion Laboratory, California Institute of Technology, under a contract with NASA; and Carnegie Mellon University. Funding for this research is provided by the Mars Technology Program, managed by the NASA Office of Space Sciences, with additional funding from the Intelligent Systems (IS) Program. The IS Program is managed by the NASA Office of Aerospace Technology.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

REFERENCES

[1] National Research Council, *New Frontiers in the Solar System: An Integrated Exploration Strategy*, The National Academies Press, 2003.

[2] NASA Office of Space Science, *Solar System Exploration Roadmap*, JPL Publication 400-1077, Jet Propulsion Laboratory, Pasadena, CA, May 2003.

[3] Bresina, J., M. Bualat, L. Edwards, R. Washington, A. Wright, "K9 Operation in May '00 Dual-Rover Field Experiment," *6th Intl. Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*, Montreal, Canada, 2001.

[4] Lawrence, G.M., J.E. Boynton, et al, "CHAMP: Camera HANDlens MicroscoPe," The 2nd MIDP Conference, Mars Instrument Development Program. JPL Technical Publication D-19508, 2000.

[5] Nesnas, I., R. Volpe, T. Estlin, H. Das, R. Petras, D. Mutz, "Toward Developing Reusable Software Components for Robotic Applications," *IEEE/RSJ International*

Conference on Intelligent Robots and Systems (IROS), 29 Oct. – 3 Nov. 2001.

[6] Pedersen, L., M. Bualat, D. Lees, D.E. Smith, R. Washington, "Integrated Demonstration of Instrument Placement, Robust Execution and Contingent Planning," *7th Intl. Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*, Nara, Japan, May 2003.

[7] Volpe, R., "Rover Functional Autonomy Development for the Mars Mobile Science Laboratory," *IEEE Aerospace Conference*, Big Sky, Montana, 8-15 March 2003.

[8] <http://telerobotics.jpl.nasa.gov/tasks/claraty>.

[9] Mars Expeditions Strategy Group, D. McCleese (chair), "Part 1: The Search for Evidence of Life on Mars," *Mars Exploration Program Analysis Group (MEPAG) Report*, July 2001.

[10] Nesnas, I.A., M. Maimone, H. Das, "Rover Maneuvering for Autonomous Vision-Based Dexterous Manipulation," *IEEE Conf. on Robotics and Automation (ICRA)*, San Francisco, CA, 24-28 April 2000.

[11] Schenker, P., T. Huntsberger, P. Pirjanian, E. Baumgartner, and E. Tunstel, "Planetary Rover Developments Supporting Mars Exploration, Sample Return and Future Human-Robotic Colonization," *Autonomous Robots* 14, 103-126, 2003.

[12] ATRV is a trademark of iRobot, <http://www.irobot.com/rwi/>.

[13] <http://www.perl.org>.

[14] Krause, Ralph, "CVS: an introduction," *Linux Journal*, Vol. 2001, Issue 87, July 2001.

[15] <http://www.gnu.org/software/make/>.

[16] <http://www.doxygen.org>.

[17] <http://www.cs.wustl.edu/~schmidt/ACE.html>.

[18] <http://cppunit.sourceforge.net/cgi-bin/moin.cgi>.

[19] Eng, Eirik, "Qt GUI Toolkit: Porting graphics to multiple platforms using a GUI toolkit," *Linux Journal*, Vol. 1996, Issue 31es, November 1996.

[20] Spasojevic, M., Satyanarayanan, M., "An empirical study of a wide-area distributed file system," *ACM Transactions on Computer Systems*, Vol. 14, Issue 2, May 1996, pp. 200-222.

[21] Urmson, C., Simmons, R., Nesnas, I., "A generic framework for robotic navigation," *IEEE Aerospace Conference*, Big Sky, Montana, 8-15 March 2003.

BIOGRAPHIES

Anne Wright graduated in 1996 from MIT with B.S. and M.Eng. degrees in Computer Science. While at MIT she worked at the Artificial Intelligence Laboratory on visual tracking, software, and systems integration for flying and walking robots, and co-founded Newton Research Labs, where she continued to work after leaving MIT. She joined code IC at Ames in 1998 to work on autonomous planetary rover testbeds, and she is currently leading the CLARAty integration effort for the K9 project.



and navigation tasks for mobile robots since 1996. Prior to working on robots, Maria was a member of the Ames Photonics research group, developing optical matrix and fourier optics processors and fiber optic microphones.

Issa A.D. Nesnas, Ph.D. is the Task Manager for the Architecture and Autonomy Research collaborative task and the principal investigator on a number of robotic research tasks at JPL. His research interests include software and hardware architectures for robotic systems and sensor-based robot control. Issa received a B.E. degree in Electrical Engineering from Manhattan College, NY, in 1991. He earned the M.S. and Ph.D. degrees in Mechanical Engineering from the University of Notre Dame, IN, in 1993 and 1995 respectively. In 1995, he joined Adept Technology Inc. as a senior project engineer. He has joined NASA at the Jet Propulsion Laboratory in 1997. At JPL he has worked on several robotic and flight projects researching autonomous sensor-based systems. He has received several Notable Organizational Value Added (NOVA) Awards and an Exceptional Achievement Award for his work at JPL. Issa holds a patent for the Impulse-based flexible parts feeder and is a member of Eta Kappa Nu and Tau Beta Pi National Honor Societies.



Clay Kunz is the lead software engineer for the K9 rover at NASA Ames. He is also the head of the math and data structures subgroup of CLARAty. He's been an employee of QSS Group, and has had his hands inside K9, at Ames since 2001, before which he spent time making robot tour guides at a start-up company in Pittsburgh, PA. Clay holds BS and MS degrees from Stanford University, and lives in San Francisco.



Maria Bualat is the project manager and lead systems engineer for the K9 rover at NASA Ames Research Center. Maria received her B.S. in Electrical Engineering from Stanford University in 1987, and her M.S. in Electrical Engineering, emphasis Control Systems, from Santa Clara University in 1992. She has been a researcher on vision