CLARAty: A Collaborative Software for Advancing Robotic Technologies

Issa A.D. Nesnas

***Abstract*— Future planetary science exploration will demand more capable and intelligent robots. Software plays a key role as it embodies the intelligence of a machine. To advance robotic technologies it becomes necessary to effectively share and reuse robotic technology implementations across projects. This calls out for a common framework for integrating robotic software to address its numerous challenges. This paper presents the CLARAty robotic software framework that was primarily developed by the Mars Technology Program for integrating advanced robotic technologies from its competed programs and their deployment on NASA's research rover fleet. We will present the multi-institutional development process and highlight some of the principles adopted in developing CLARAty. We will summarize both technical and non-technical challenges and close with an example of the successful sharing of robotic software infrastructure and component technologies among institutions.**

I. Introduction

THE development of intelligent robotic systems is hard because of the multi-disciplinary nature of its constituent technologies and the complexity of their integration. The process of bringing intelligence to a robot requires the effective melding of sensing, reasoning, and motion technologies. As such, software plays a key role as it is the medium that embodies intelligence in a machine [1].

Nevertheless, within the NASA robotics community, and to a large extent within the research community, the majority of robotic software is designed and built from scratch for each new robot. To date, it may have been easier and more cost effective to do so. However, as the need for more advanced robotic capabilities for future science missions increases, it becomes necessary to leverage prior robotic technology investments.

Effective leveraging of software from multiple sources requires the addressing of both architectural and integration issues. Therefore, it is necessary to have a dedicated and focused effort that addresses current needs but one that is also forward looking to support future advances in robotic technologies. A natural outcome of such an effort would be a common framework for the development, deployment and use of robotic technologies across institutions.

Advancing state-of-the-art in robotic technology involves the effective sharing of software across institutions. Without such effort, disparate robotic efforts will be hindered by the need to reinvent and re-implement capabilities that exist in other systems or have existed in the past but have been abandoned after the project has been disbanded and its developers dispersed.

To that end, the NASA Science Directorate, through its Mars Technology Program, has been developing the CLARAty robotic software framework [2][3]. The main objective is to enable the effective leveraging of robotic software capabilities among participants to achieve a higher level of robot intelligence. Program participants are distributed among NASA centers, universities and industry.

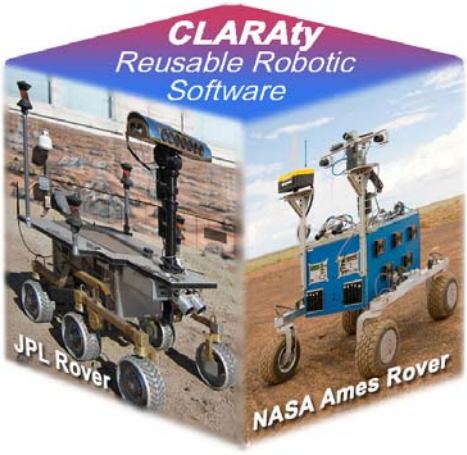

Figure 1: Example of two platforms that run CLARAty: the Rocky 8 rover at JPL (left) and the K10 rover at NASA Ames

CLARAty stands for Coupled-Layer Architecture for Robotic Autonomy. It is a generic framework for reusable robotic software that facilitates the integration and deployment of advanced technologies onto NASA's robotic platforms. CLARAty is a multi-institutional collaboration with software developers at four institutions: the Jet Propulsion Laboratory (JPL), NASA Ames Research Center, Carnegie Mellon and University of Minnesota.

CLARAty has been deployed on multiple platforms over the past several years. Figure 1 shows two such platforms: the Rocky 8 rover at JPL and the K10 rover at NASA Ames. Other robots that run CLARAty include the Rocky 7, FIDO and K9 rovers and a commercial-off-the-shelf ATRV rover used by Carnegie Mellon and the University of Minnesota. In addition to these hardware deployments, CLARAty has also been adapted to the ROAMS high-fidelity simulator [4]. Both deployments were used by the Mars Science Laboratory Focused Technology Program for the validation of advanced robotic technologies for consideration by the flight missions [5].

This paper provides an overview of the development process and a description of the architecture. We also summarize the challenges of developing reusable robotic software and highlight some of the results that demonstrated higher-level robot intelligence through this collaborative process.

Because this framework was designed to be generic, it can

also support legged and aerial platforms which are of interest to the Lunar Exploration and Solar Systems Exploration Programs respectively.

II. Related Efforts

The idea of developing a common software framework for robotics dates back to two decades. Several efforts including ones led by NASA ([6][7] and more recently [8]) recognized the importance of a disciplined approach to developing, integrating and validating robotic technologies. However, early efforts faced severe challenges and had limited success. Nevertheless, advances in both computational hardware and the software engineering are now enabling renewed efforts toward this goal.

Such efforts can be divided into two categories: (a) ones that focus on the mechanisms for information sharing independent of the domain knowledge, and (b) ones that use robotic domain knowledge to drive the design.

Among the efforts that focused on the mechanisms for information sharing are software component technologies [9]. While such technologies have a larger applicability because of their general nature, the complexity of the software engineering, the maturity and scalability of the tools, coupled with their high-cost and the readiness of the community to adopt them hindered the wide spread acceptance for standardizing robotic architectures. Some of the first examples of such architectures included Chimera [9], ControlShell [11], and the CORBA-based Mobility from IRobot. Without a focus on domain models, such generic software tools, even though well-intentioned and designed, exposed too much software engineering and became too general and abstract for roboticists who were more concerned with furthering their robotic technologies than with software. Another effort that focuses on the form of the interface and less on the content is the Foundation for Intelligent Physical Agents (FIPA) [12].

In the second category, efforts that used robotic domain engineering focused on various aspects. These resulted in solutions with different emphasis. CLARAty falls within this latter category.

In this category, some architectures focused on spatial or temporal hierarchies [6], while others focused on behavioral hierarchies [13]. More recently, the focus has been on decompositions between decisional and functional layers [2][14]. Other architectures focused on the kinematics and dynamics domain [15] or on hard real-time services [16].

Some architectures emphasized standardizing interfaces to robot hardware and control processes. Probably the most visible effort is the Joint Architecture for Unmanned Systems [17], which aims at providing standardized message passing interfaces for all of the military's unmanned vehicles. JAUS was initially developed by the Department of Defense to ensure interoperability among a family of Unmanned Ground Vehicles. Later it was extended to Aerial platforms. Similar to CLARAty, JAUS defines interfaces that are independent of the integrated technology or the specific hardware platforms. While the goals of JAUS are similar to those of CLARAty, the approaches have significant differences. The JAUS architecture uses a single-level message-set, while CLARAty uses a multi-level abstraction model.

Another effort that falls in this category is Player/Stage [18], which provides abstractions for robotic devices. It is based on a client/server model that uses socket-based communications, which requires a serialization scheme and incurs a significant cost for resource-constrained robots. Additionally, the current Player abstractions only address a limited set of capabilities primarily geared towards controlling commercial-off-the-shelf robots with simple mobility mechanisms.

The former category led to solutions that were too general and complex for robotic researchers to adopt. The latter category resulted in domain specific solutions to problems with limited levels of software reuse within that domain.

In addition to these efforts, during the last very few years, a number of worldwide initiatives have been undertaken. In 2002, Intel Corporation established the Robotics Engineering Task Force [24], which was a coalition of industry, academic and government participants. Modelled after the Internet Engineering Task Force, the RETF's primary goal was to specify interoperable software interfaces for mobile robots. A session was dedicated to this effort at the 2003 IROS conference. The IEEE Robotic and Automation Society has organized two full-day workshops [20][21] on Software Development and Integration in Robotics at its 2005 and 2007 international conferences in Barcelona and Rome respectively. Supported by the Japan Robot Association, the Object Management Group established a Robotics Special Interest Group [22], which aims at defining a robotics domain architecture based on OMG's standards.

The importance and challenges of this emerging field are clearly demonstrated by the multitude of efforts within the military, space and research communities that are striving to establish software standards and frameworks for facilitating the integration of robotic technologies to enable faster advances in robotic intelligence.

III. The Development Process

Driven by the desire to deploy more capable robots for planetary science exploration, NASA identified the need for establishing an integration framework for its robotic technologies that are being developed within the Mars Technology Program. At NASA, we were fortunate to have the programmatic support and a critical mass in advanced robotic development to enable us to make a contribution to the field of robotic software integration.

The development of the CLARAty framework was possible for three reasons. First and foremost, it was driven by the need to integrate technologies developed by external and internal participants of the two competed rounds of MTP's NASA Research Announcements over the past six years. This also opened the door for the integration of technologies from other NASA programs such as the Intelligent Systems Program, which was part of the Computing, Information and

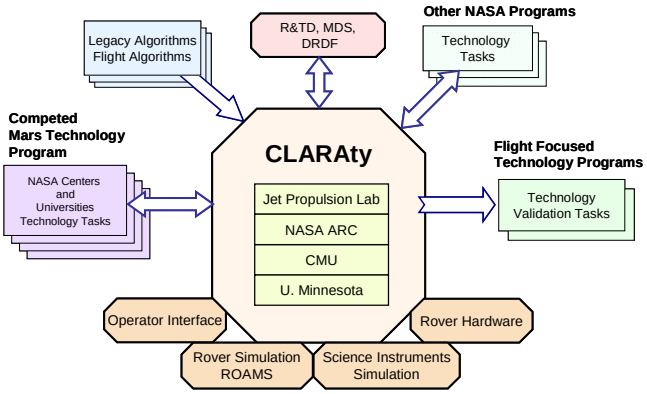

Figure 2: The inflow of technology algorithms from multiple NASA programs and flight projects and the outflow of algorithms deployed on research rovers for formal validation

Communications Technology Program (CICT), and other internal programs. Second it addressed a programmatic need to, more cost effectively, share robotic technologies among centers and reduce the overall costs of robotic software development and maintenance for NASA's heterogeneous research rover fleet. Third, it enabled the validation of component technologies for consideration into flight projects. With an infrastructure that enabled the validation of robotic technologies, legacy algorithms from previous Mars missions have been integrated into CLARAty for formal validation. As a consequence, these technologies serve as a baseline to compare recent advances against. That not only helps establish a baseline, but it also enables the conduction of comparison experiments under controlled environments. CLARAty supported experiments with a high-fidelity rover and terrain simulator [4] and with research rovers operating in the JPL outdoor Mars Yard. Examples of legacy algorithms that have been integrated into CLARAty include the Sojourner rover pose estimation algorithm [23], the MER vision-based obstacle avoidance algorithm [24], and the MER visual target tracking algorithm [25], and the MER visual odometry [26]. Figure 2 shows the multi-center CLARAty development and its close collaboration with other tasks that support the rover hardware fleet, the rover and science instruments simulations, and the science operator interface [27]. Technologies flow in from multiple external and internal programs and the flow out to independent formal validation tasks.

From its onset, CLARAty was setup as a collaborative effort to bring domain experts from leading institutions to develop a *common framework* for space robots. That group formed the core development team that focused on developing reusable robotic software for supporting the integration of advanced robotic technologies. Unlike some of the efforts mentioned in the previous section, CLARAty was grounded by the need to deploy the framework early to external technology developers in the program to integrate their products onto the rover fleet for formal validation. As a consequence, the overall process comprised the design, development, integration, deployment, validation, and capture of lessons learned to feed the next cycle of development. This iterative development process enabled us to mature the design by capturing lessons learned from deployed systems.

The process of developing CLARAty was made easier by starting from existing implementations of legacy systems for each of the rovers. We were lucky to have full realization of software for the Rocky 7, Rocky 8, K9 and FIDO rovers; all of which had software that was developed by independent teams. We then did a commonality/variability analysis [30] to define the common abstract models for these systems. Later, we adapted these abstract models back to these platforms and tested them on existing hardware. Our approach can be summarized as follows:

- Capture **requirements** from domain experts at multiple institutions
- Use **global perspective** across domains (motion, vision, estimation, navigation)
- Identify **recurring patterns** and **common infrastructure** therein
- Use **domain experts** to guide design
- Define **proper interfaces** for each subsystem
- Develop a **generic framework** to support various implementations
- Adapt **legacy implementations** to validate framework
- **Encapsulate** when re-factoring is not feasible or affordable
- Develop **regression tests** where feasible
- **Test** on multiple robotic platforms and **study limitations**
- **Feed** learned experience **back** into the design
- **Review** and **update** to address limitations

After several iterations one hopes to have achieved a truly reusable infrastructure for that class of robots.

The ultimate goal is to build a robotic software system that is reusable across robots and that supports different operational scenarios. As a result the software has to be stable against hardware variability, flexible enough to accommodate the requirements of a continuously evolving application field. The software also has to be easy to understand and maintain.

IV. The Architecture

CLARAty adopts a layered architectural model, with each layer unified around a different programming paradigm. CLARAty decomposes robotic software into two layers: a decision layer and a functional layer. The decision layer uses a declarative programming paradigm, which has been the focus of research efforts within the artificial intelligence community. The functional layer uses a multi-abstraction model based on procedural programming paradigm, which has been dominant within the robotics community. These two programming paradigms are quite different for building robotic intelligence.

The decision layer adopts a declarative programming paradigm where the programmer explicitly describes the activities, models, and constraints but does not provide any program logic (sequences, conditionals, and loops) that describes the order of execution. The program logic is automatically generated and updated by a search-engine that

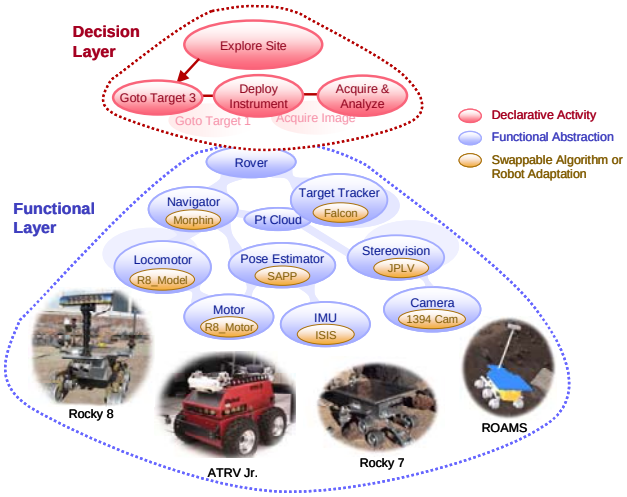

Figure 3: The overall CLARAty architecture with a declarative-based decision layer and procedural-based functional layer

examines all constraints and maintains a plan to order activities without violating these constraints [28].

Conversely, the functional layer adopts a procedural programming paradigm, which readily provides the program logic that contains the order of execution using activity sequencing, conditionals, loops, and concurrent activities. The execution is only altered through conditionals, exceptions, and dynamic binding. While declarative programming has larger flexibility in ordering activities than the procedural programming, it requires computational resources to generate the program logic and requires explicit constraints on all activities.

Figure 3 shows the two-layer architecture with the decision and functional layers. The functional layer uses a design based on object models. Abstract models of various capabilities are defined with interfaces that link the different components in the system. These models use different technologies and algorithms to implement the capabilities. This modular approach enables the evaluation of different technologies without having to re-architect the entire system. For example, the generic navigator provides an abstract model for navigating a rover in rough terrain. Navigation algorithms with different algorithms have been adapted to CLARAty including the one that was used on the MER rovers on Mars [24].

The design of the functional layer is governed by a number of principles [29] that evolved from the iterative development process described in the previous section. We highlight some of these principles below:

1. Separation of the intent from the implementation (the "what" from the "how")
2. Generalization and stabilization of interfaces by using complex data structures
3. Separation of generic runtime models from platforms for specific ones
4. Exposure of stable behaviors and interfaces while encapsulating state and runtime models

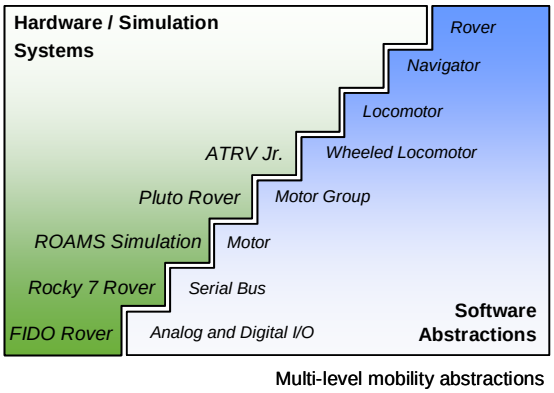

Figure 4: Adaptation of devices at different levels of abstraction

5. Enabling integration of algorithms and adaptations of devices at different levels of abstractions
6. Employing a cross-domain analysis to maximize reuse of data structures and optimize data flow
7. Definition of common policies for cross-cutting themes such as time, uncertainty, and coordinate transformations
8. Unification of the mechanism model representation
9. Separation of the mechanism model from the control
10. Separation of logical and physical hierarchies for devices

Figure 4 illustrates the 5th principle. It shows the abstraction levels of a locomotion example where access to hardware or simulation occurs at various levels. At the lowest levels, the control software interfaces to hardware or simulation using low-level analog and digital I/O signals. A higher level would be to interface through a serial bus where information is exchanged through formatted data packets as opposed to toggling I/O signals. At an even higher level, the software interfaces to a motor adaptation that understands motor commands. Higher levels include an interface to a group of coordinated motors, a wheeled locomotor, or a general locomotor. The level at which the interface to hardware or simulation is made depends on the openness of the hardware or the level of fidelity of the simulator. For example, access to a custom-built robot often occurs at lower levels while access to a commercial-of-the-shelf robot occurs at a higher level because the manufacturer may only provide an interface to control the robot's motion and not individual wheels. In this case, the concept of a motor is hidden in a layer that is inaccessible to the user. Such a system will have adaptations at the locomotor level providing only access to the locomotor interface as opposed to the motor or I/O level interfaces.

V. Challenges

The primary challenge in developing reusable robotic software is software instability, which results from the variability in robotic hardware, variability in the operating environment and variability in the operational tasks. In addition to technical challenges, developing unified robotic software has its share of programmatic challenges.

Hardware variability results from (a) differences in robotic

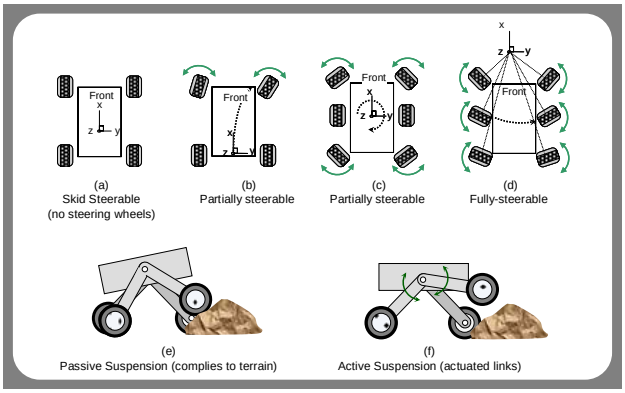

Figure 6: Different mechanisms for wheeled robots

hardware architectures, (b) differences in the hardware components used by various systems, (c) differences in sensor configurations for a given system or application, and (d) differences in the mechanisms that generate motion in a robot.

To illustrate the extent of the latter, consider the variations within the class of wheeled rovers shown in Figure 5. Different mechanisms exhibit different capabilities, which have major implications on how the robots move and act. Controlling wheeled robots is very different from controlling legged platforms. Even among wheeled robots, fully-steerable (omni-directional) rovers can move laterally (crab) while partially-steerable (car-like) robots can achieve the same result only via a parallel parking maneuver Figure 5 (a) to (d)). Mobile robots with passive suspension conform to the terrain with no control over their tilt while those with active suspension have control over their tilt (Fig. 5(e) and (f)). Robotic manipulators have similar nuances depending on the number of degrees of freedom they have and their joint configurations.

In addition to hardware variability, software instability emanates from (a) software complexity, (b) advanced algorithm integration, (c) architectural mismatches, (d) variability of tasks and tools (e.g. operating systems and software development tools).

Beyond technical challenges, there are non-technical challenges in developing and sharing a common robotic framework. First and foremost are the Export Control regulations that govern different software modules. In this paper, we described a generic software framework that enabled us to integrate different technical solutions including ones chosen for flight missions. While we have demonstrated software interoperability of these algorithms on real platforms, access to different implementations may not always be readily possible. Access is governed by a complex set of policies to ensure compliance with International Traffic and Arms Regulations (ITAR) laws. As a result, we had to place sophisticated mechanisms to control access based on an individual's credentials.

In addition to ITAR and Commerce restrictions, there are Intellectual Property restrictions associated with software developed by multiple institutions. From that perspective, CLARAty is divided into two categories: (1) infrastructure modules that have shared ownership among the centers and is governed by a single multi-center intellectual property document, and (2) individual technologies, each with its own Intellectual Property document.

Beyond such challenges, there are practical challenges of developing software remotely across institutions. These include challenges in testing and validating new features without shared hardware platforms.

Despite these challenges, progress in this field has already yielded several rewards for multiple institutions. Today robotic software infrastructure and algorithms are being successfully shared and validated across half a dozen institutions. Such sharing enables the distributed teams to leverage each other's developments leading to more efficient development and integration of advanced algorithms. In the next section, we will describe one such example in some detail.

VI. Results

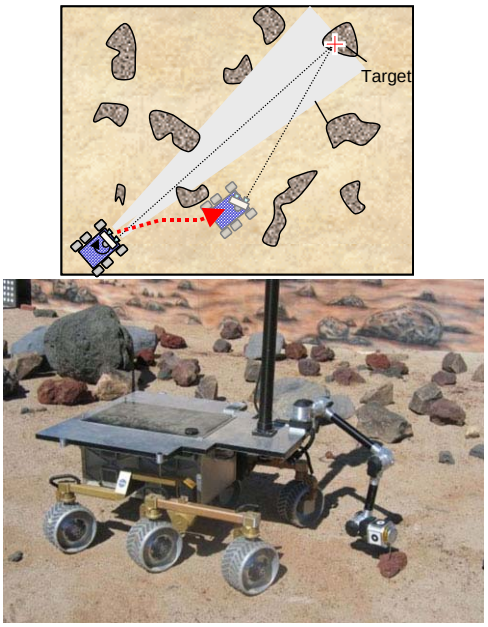

Figure 5: Rocky 8 executing single-cycle instrument placement

The CLARAty framework enabled development and integration of capabilities from multiple institutions. One notable example is the single-cycle instrument placement (SCIP) capability.

Algorithms for SCIP were developed at various institutions and have been integrated into CLARAty and deployed on both the Rocky 8 rover at JPL and the K9 rover at NASA Ames. Figure 6 shows the SCIP capability, where a scientist designates a target from the science panoramic cameras at a distance of 10 rover lengths away. With its built-in autonomous capability, the rover would then drive while continuously tracking the target using the rover stereo cameras. While tracking the target, the rover would use its body cameras to detect and avoid obstacles along the way. As the rover gets closer to the target, the SCIP algorithm will switch the tracking of the target from the narrow field-of-view

(FOV) pancams to the wider FOV navigation cameras (navcams) that are mounted on the same articulated mast. The rover will continue tracking the target after it has been handed-off from one camera pair to another. As the rover comes up to the target rock, another hand-off operation from the navcams to the body hazard cameras (hazcams) is carried out. This hand-off operation is more challenging because the body cameras has three times wider FOV compared to the navcams. They also have a significantly different view point of the target. During its final steps, the rover positions the rover base, continues to track the target from the hazard cameras and places the rover such that the target is within the arm's reach. The surface normal of the target is computed during this final approach phase to determine the rover's final placement. Once in its final position, the arm is deployed and the end effector motion is sensed as the rover touches the designated target. The science instrument mounted at the end of the arm acquires the data and the rover simulates a downlink to Earth.

Such capabilities require the integration of a number of technologies such as: motion control and coordination of the mobility platform and the arm, stereo vision, visual target tracking, target hand-off, pose estimation, path planning, navigation and obstacle avoidance.

For each component, a number of different technologies were explored. For example, different techniques were tested for the visual target tracking included image-based tracking, shape-based tracking, as well as hybrid methods. We also explored different technologies for pose estimation, which included wheel odometry methods and ones that combined wheel and visual odometry. Multiple navigation algorithms have been used. The component technologies were provided by JPL, NASA Ames, Carnegie Mellon and University of Minnesota.

VII. Conclusion

There are many challenges to developing reusable robotic software both technical and non-technical. However, the scope of software development that is necessary to build intelligent software exceeds the resources that are available for most projects today. As a result, developing an infrastructure for sharing and leveraging each other's software proved reasonably effective within our small team of a dozen institutions. CLARAty is on the verge of makings its first public release and lesson learned from that experience will be published in a future publication.

VIII. Acknowledgements

The author would like to acknowledge the contributions of current and former members of the CLARAty team. In particular: Richard Volpe, Dan Clouse, Antonio Diaz-Calderon, Tara Estlin, Daniel Gaines, Won Kim, Richard Madison, Michael McHenry, Hari Nayar, Richard Petras, Mihail Pivtoraiko, Gregg Rabideau, I-Hsiang Shu, from the Jet Propulsion Laboratory; Clayton Kunz, Lorenzo Fluckeiger, Randy Sargent, Hans Utz, and Anne Wright from NASA Ames Research Center; Reid Simmons, David Apfelbaum, Kam Lasater, Nik Melchoir, and Chris Urmson from Carnegie Mellon; and Stergios Roumeliotis, Anastasios Mourikis, and Nikolas Trawny from the University of Minnesota. The author would like to acknowledge the numerous contributors from universities and NASA centers who have provided algorithms to CLARAty. The author would also like to thank NASA's Mars Technology Program for their vision and support: David Lavery, Samad Hayati, Paul Schenker, Richard Volpe, and Gabriel Udomkesmalee. The work described in this chapter was carried out at the Jet Propulsion Laboratory, California Institute of Technology, NASA Ames Research Center, Carnegie Mellon, and University of Minnesota under a contract to the National Aeronautics and Space Administration.

References

[1] D. Brugali (ed.) "Software Engineering for Experimental Robotics", *Springer Tracts on Advanced Robotics*, March 2007.

[2] I.A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. Shu, and D. Apfelbaum, "CLARAty: Challenges and Steps Toward Reusable Robotic Software," *International Journal of Advanced Robotic Systems*, Vol. 3, No. 1, pp. 023-030, 2006.

[3] R. Volpe, I.A.D. Nesnas, T. Estlin, D. Mutz, R. Petras, H. Das, "The CLARAty Architecture for Robotic Autonomy." Proceedings of the 2001 *IEEE Aerospace Conference, Big Sky Montana*, March 10-17 2001.

[4] A. Jain and et.al., Recent developments in the roams planetary rover simulation environment, *IEEE Aerospace Conference* (Big Sky, Montana), 2004.

[5] R. Volpe, "Rover Technology Development and Mission Infusion Beyond MER," 2005 IEEE Aerospace Conference, Big Sky, Montana, March 6-11, 2005.

[6] J. Albus, H. McCain, and R. Lumia, NASA/NBS standard reference model for telerobot control system architecture (NASREM), NBS Technical Note 1235, National Bureau of Standards, Gaithersburg, Maryland, July 1987.

[7] R. Volpe, J. Balaram, T. Ohm, and R. Ivlev, "The Rocky 7 Mars Rover Prototype," *IEEE / RSJ International Conference on Intelligent Robots and Systems (IROS)*, Osaka, Japan, November 4-8, 1996.

[8] A. Bradley, S. Dubowsky, R. Quinn & N. Marzwell. Enabling Interoperable Space Robots with the Joint Technical Architecture for Robotic Systems (JTARS), *International Symposium on Artificial Intelligence, Robotics and Automation in Space (I-SAIRAS)*, Munich, Germany, September 2005.

[9] Component software - beyond object-oriented programming, Addison-Wesley / ACM Press, 2002.

[10] D.B. Stewart and P. Khosla. The Chimera Methodology: Designing Dynamically Reconfigurable and Reusable Real-Time Software using Port-Based Objects. *International Journal of Software Engineering and Knowledge Engineering*, Vol. 6, No. 2, June, 1996, pp. 249-277.

[11] G. Pardo-Castellote S. Schneider, V. Chen and H. Wang. Controlshell: A software architecture for complex electromechanical systems. *Int'l Journal of Robotics Research*, 17(4), April 1988

[12] FIPA (2005). Foundation for Intelligent Physical Agents, http://www.fipa.org/

[13] R.A. Brooks (1991) Intelligence Without Reason, Proceedings of the 12th Int. Joint Conference on Artificial Intelligence, Sidney 1991

[14] R. Alami, R. Chautila, S. Fleury, M. Ghallab, and F. Ingrand, An architecture for autonomy, The International Journal of Robotics Research 17 (1998), no. 4.

[15] C. Kapoor, D. Tesar, D. (1998). A Reusable Operational Software Architecture for Advanced Robotics, *CSIMIFToMM Symposium on theory and Practice of Robotsand Manipulators*, Paris, France.

[16] OROCOS (2005) http://www.orocos.org/

[17] JAUS (2005). Joint Architecture for Unmanned Systems Ref. Architecture, Ver. 3.0, http://www. jauswg.org/.

[18] Gerkey, B.; Vaughan, B.; & Howard, A. (2003). The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems,

International Conference on Advanced Robotics, pages 317-323, Portugal.
[19] Robotics, Engineering Task Force http://www.robo-etf.org.
[20] IEEE ICRA2005 Workshop on Software Development and Integration in robotics (SDIR2005), Barcelona, Spain April, 18 2005 http://robotics.unibg.it/tcprog/sdir2005/
[21] IEEE ICRA2007 Workshop on Software Development and Integration in robotics (SDIR2007), Rome, Italy April, 14 2007 http://robotics.unibg.it/tcprog/sdir2005/
[22] Robotics Domain Special Interest Group, http://www.omg.org/news/releases/pr2005/02-17-05.htm.
[23] A. Mishkin, J. Morrison, T. Nguyen, H. Stone, B. Cooper, & B. Wilcox, (1998) Experiences with Operations and Autonomy of the Mars Pathfinder Microrover, *IEEE Aerospace Conference*, Colorado.
[24] M. W. Maimone, P. C. Leger, J. J. Biesiadecki, "Overview of the Mars Exploration Rovers' Autonomous Mobility and Vision Capabilities," *IEEE International Conference on Robotics and Automation (ICRA)*, Space Robotics Workshop, Roma, Italy, 14 April 2007.
[25] W. S. Kim, A. I. Ansar, R.D. Steele, K.S. Ali, I.A. Nesnas, "Rover-Based Visual Target Tracking Validation and Mission Infusion," *AIAA Conf.*, Space 2005, Aug. 2005.
[26] M.W. Maimone, Y. Cheng, L. Matthies, “Two Years of Visual Odometry on the Mars Exploration Rovers,” *Journal of Field Robotics*, Volume 24 number 3, special issue on Space Robotics, March 2007, 169 - 186.
[27] J. S. Norris, M. W. Powell, J. M. Fox, K. J. Rabe, I. Shu, "Science Operations Interfaces for Mars Surface Exploration," 2005 *IEEE Conference on Systems, Man, and Cybernetics*, October 15-17, Big Island, HI., October 15, 2005.
[28] T. Estlin, D. Gaines, C. Chouinard, F. Fisher, R. Castano, M. Judd, R. Anderson, and , I. Nesnas, "Enabling Autonomous Rover Science Through Dynamic Planning and Scheduling," *Proceedings of the 2005 IEEE Aerospace Conference*, Big Sky, Montanna, March 2005.
[29] I. Nesnas, The CLARAty Project: Coping with Hardware and Software Heterogeneity, In Software Engineering for Experimental Robotics, D. Brugali (Ed.) , Springer STAR 2007.
[30] J. Coplien, D. Hoffman, and D. Weiss. Commonality and variability in software engineering. IEEE Software, 15(6), 1998.