AUTONOMOUS ROBOTICS TOOLBOX

Erick Dupuis, Pierre Allard, Régent L'Archevêque

Canadian Space Agency, Space Technologies
6767 route de l'Aéroport, St-Hubert (Qc), J3Y 8Y9, Canada
Email:firstname.lastname@space.gc.ca

ABSTRACT

Autonomy is required to support current and future space robotics missions. Planetary exploration robots and unmanned reusable space vehicles will require a high level of autonomy to perform tasks more efficiently.

Over the last 5 years, the Canadian Space Agency has designed, implemented and tested different autonomy techniques on typical autonomous robotics scenarios. The approaches that were tested include Finite State Machines (FSM), Hierarchical Task Networks (HTN) and Goal Decomposition Hierarchies (GDH).

The ARGO Cortex Toolbox merges some of these techniques and allows the implementation of hierarchies of finite state machines. It allows state machines to be designed modularly in order to be reused in different contexts. This feature permits the creation of FSM Libraries. The Cortex Panel provides features to design, manage, generate code, execute, and to monitor execution in real-time or offline. The current release supports Java and code generation. The code generated can then be integrated into the specific framework depending of the application.

1 INTRODUCTION

Over the last few decades, robots have played an increasingly important role in the success of space missions. The Shuttle Remote Manipulator System (also known as Canadarm) on the Space Shuttle has enabled the on-orbit maintenance of precious space assets such as the Hubble Space Telescope. On the International Space Station (ISS), the Canadarm 2 has been a crucial element in all construction activities. Its sibling, Dextre, will be essential to the maintenance of the ISS. In the context of planetary exploration, robotics has also played a central role on most landed missions. The rovers "Spirit" and "Opportunity" are vibrant examples of how robots can allow scientists to make discoveries that would be otherwise impossible.
One of the current drawbacks of space robots is that their operation is very human-intensive. On manned platforms such as the Space Shuttle and the ISS, the baseline for operations requires the involvement of an astronaut for control and monitoring and the support of ground support personnel to monitor the operations in real-time. The advent of ground control for the Canadarm2 on the ISS has freed up the crew from having to perform all robotic operations. Ground-based operators now have the capability to uplink individual commands for automatic execution on the ISS.

In the case of robotic planetary exploration missions, the constraints associated with the communications between the robot on the surface of a remote planet and Earth-based operators are much more stringent. For example, in the case of Mars exploration missions, the round-trip time delays are on the order of 10 to 40 minutes. Furthermore, it is not uncommon to have situations where there are communication windows of only one hour over a period of 12 hours. In addition, the robot must operate in an environment that is not known a-priori. Environment models are built on the fly and are made available to the operations teams every cycle.

So far very little is implemented in terms of autonomous decision-making capability. Operations are based on pre-planned, pre-verified command scripts. When situations requiring some sort of decision are encountered, the robot must usually stop and wait for a human operator to intervene. For example, in the case of the rovers "Spirit" and "Opportunity", once an interesting geological feature has been identified, it takes three command cycles (of 12 hours each) to go apply an instrument to it [1]. The scientific return on investment is therefore severely limited by the lack of on-board autonomy capability.

Several architectures have been developed to address the issues associated with ground control of space-based robots. However, since the usual mode of operation for space robotics in the past has been tele-operation with direct operator control or supervision, most of the approaches have not focused on the implementation of autonomy. In the late 1990's, the Canadian Space Agency and their industrial partner, MD Robotics, have developed the Intelligent, Interactive Robotic Operations (IIRO) framework

Proc. of 'The 8th International Symposium on Artifical Intelligence, Robotics and Automation in Space - iSAIRAS', Munich, Germany.
5-8 September 2005, (ESA SP-603, August 2005)

[2][3], which allowed the teleoperation of remote equipment in operational settings. The Remote Operations with Supervised Autonomy (ROSA) architecture was a follow-on to IIRO and addressed the issue of scripted control and basic autonomy requirements [4]. ROSA has been used as the basis for the development of the ground control station for the robotic elements on the Orbital Express satellite-servicing mission [5].

Similar architectures were developed in Europe at the same time. The Modular Architecture for Robot Control (MARCO) developed by DLR addresses similar issues in the context of tele-operation and some aspects of scripted play-back [6]. The MARCO architecture and its relatives have been used on several missions including ROTEX and ETS-7. Two other architectures were also developed under the leadership of the European Space Agency: FAMOUS and DREAMS also concentrated on the issues associated with the teleoperation of robots in space. In both cases, special attention was dedicated to the issues surrounding planning, verification and execution of command sequences.

Despite the wealth of research in autonomous robotics and in control architectures for space robots, relatively little has been done to address specifically the needs of autonomous space robots. NASA/JPL have been developing/proposing two different architectures for applications with a higher degree of autonomy in the last few years: CAMPOUT and CLARAty. CAMPOUT [7] is a control architecture for the real time operation of multiple rovers. CLARAty [8][9] is a proprietary architecture that is becoming a requirement for many missions. The main goal of CLARAty is to provide a systematic framework for treating all the different vehicles/robots/instruments involved in Mars missions. CLARAty is object oriented and according to the publications provides a high degree of reusability.

In parallel with these efforts, the Canadian Space Agency has been developing the Autonomous Robotics and Ground Operations (*ARGO*) software suite. *ARGO* provides a framework for the integration of the space operations process from planning to post-flight analysis. The objective of *ARGO* is to reduce operational costs and increase efficiency by providing operator aids and permitting the implementation of a level of autonomy appropriate to the application.

The target applications of the *ARGO* framework [10] cover the full spectrum of autonomy from supervisory control such as might be expected for ISS robotics to more autonomous operations such as would be encountered in planetary exploration missions. It also covers the full range of space robotic applications from orbital manipulators to planetary exploration rovers. One of the important features of the *ARGO* framework is that it does not provide a universal architecture for ground control and autonomy. Instead, *ARGO* provides a set of toolboxes that can be assembled in a variety of manners depending on the application and its requirements. To facilitate the re-use of software, the design is modular and portable to the maximum extent possible

Several toolboxes already exist within the *ARGO* framework. The central element of the *ARGO* framework is the *Cortex Toolbox*, which provides a set of tools to implement on-board autonomy software based on the concept of hierarchical finite state machines. The *Cortex Toolbox* allows an operator to graphically generate the behaviours to be implemented on the remote system. It automatically generates the code to be uploaded and it can be used to debug and monitor the execution of the autonomy software on-line and off-line.

2 CONCEPT OF OPERATION

The basic premise behind the design of *ARGO* is the integration into a single environment of the operations process from planning through verification to execution and post-flight analysis. *ARGO* provides a set of tools that can be connected in context-dependent configurations. For example, to plan and verify command sequences, the command and telemetry interfaces can be connected to a virtual environment composed of a simulation model of the system to be controlled along with a graphical rendering of the system and its environment. At run-time, the same command and telemetry interfaces get connected seamlessly to the real system in space to upload the pre-verified command scripts. After execution, it is possible to connect the telemetry interface to the logged telemetry files to conduct off-line post-flight analyses.

One key feature of *ARGO* is the capability to implement varying levels of autonomy as appropriate for the target application. This is usually dictated by the quality of the information being fed back to the operator. Autonomy is not required when the operator has adequate, up-to-date information and the ability to intervene in a timely manner. However, if the operator only receives out-of-date information or loses the ability to intervene, then he is not capable of making timely decisions and some amount of local decision-making capability is required at the remote site. Factors that can influence the level of autonomy required include long time delays, low communication bandwidths, intermittent windows of communication, poor situational awareness and a dynamic environment.

For example, in the presence of communication links with low latency, high bandwidth and high reliability, it is possible to maintain the operator in the loop for every decision. Primitive commands can be sent to the remote robot one at a time and confirmations can be requested from the operator before any command is executed. In such a case, the operator can intervene and over-ride the robot in case of anomaly.

In contrast, a rover on the surface of another planet is subject to communications with long delays, narrow bandwidth and frequent blackouts. The operator has relatively poor situational awareness because of the bandwidth limitations and no ability to intervene in a timely manner because of the long delays and frequent blackouts. The environment is unstructured and, in some cases, could even be unknown to the operator. In this case, it is preferable to implement some autonomous decision-making capability. Relying on the operator for every decision will result in long idle times between communication windows while the robot is waiting for instructions.

To implement such a variety of levels of autonomy, the *ARGO* framework makes use of concepts such as command scripts and autonomous behaviours. Command scripts are files in which sequences of commands are recorded for automatic execution. The scripts are built using the finite state machine formalism, which is a convenient powerful way to represent logical rules. Commands are represented as discrete states that are linked by event-driven state transitions. In this context, states represent individual actions to be performed by the robot and transitions are events that cause the script to leave its current state to move into the next one. Typical state transitions can include external events such as sensor values and operator inputs, as well as internal events such as completion of the previous command or failure to complete it. The finite state machine formalism allows the implementation of decision-based branching as well as loops in the script. The simplest incarnation of a command script is a linear series of primitive commands for the robot to execute from start to finish.
Autonomous behaviours are built using the same methodology as scripts. In fact, behaviours can be seen as sub-scripts that can be invoked to handle some pre-determined conditions that the robot is expected to face during its mission. Whereas scripts are specific to one particular scenario, behaviours are libraries of action sequences to be undertaken by the robot under triggering conditions. Behaviours, therefore, provide the robot with some amount of reactive autonomy to make decision and take action on a determined set of events or conditions.

Behaviours can be hierarchical in nature: i.e. the states in the finite state machine representing a behaviour can themselves be behaviours. Behaviours provide the capability for operators to generate much more compact command scripts since complex operation sequences can be encapsulated in a single command that can be invoked at different times in a script or even by another behaviour.

The fact that *ARGO* treats behaviours in the same manner as command scripts allows the operator to program, verify and uplink autonomous behaviours in the same development environment that is used for operations planning. Thus, *ARGO* truly provides an integrated environment for all operations-related issues from design and testing of autonomous behaviours to planning, verification and execution of command scripts.

3 THE *CORTEX* TOOLBOX

The central element of the *ARGO* framework is the *Cortex Toolbox*, which is used to implement command scripts and sets of reactive behaviours. *Cortex* has been developed in light of the fact that the development of such behaviour sets rapidly becomes labour intensive even for relatively simple systems when using low level programming languages, thus making reusability very difficult if not impossible. *Cortex* is based on the Finite State Machine (FSM) formalism, which provides a higher-level way of creating, modifying, debugging, and monitoring such reactive autonomy engines. Some advantages of this representation are its intuitiveness and ease with which it can be graphically constructed and monitored by human operators.

The concept of hierarchical FSM allows a high-level FSM to invoke a lower-level FSM. This provides the capability to implement hierarchical task decomposition from a high-level task into a sequence of lower-level tasks. If the FSM is implemented in a modular fashion, it allows the implementation of the concept of libraries that provide the operator with the re-use of FSM from one application to another.

In general, FSMs are used to represent a system using a *finite* number of configurations, called *states*, defined by the system parameters or its current actions. In the FSM shown in *Figure 1*, the states are **Starting_Motion**, **Turning,** and **Stopping_Motion.** In this case, actions are defined in state, and they occur during state entry, re-entry and exit. For example, in *Figure 1*, when entering the **Turning** state, the current robot azimuth angle could be recorded to serve as a starting point for the destination angle computation.

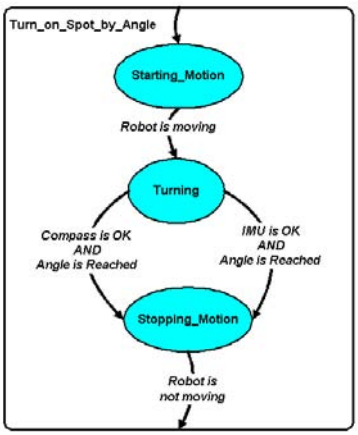

Figure 1 - A Simple Finite State Machine

The system can *transition* from one state to another based on its current state, conditions and outside events. Conditions on transitions are often referred to as *Guards*, and are implemented as statements that can be evaluated as being either true or false. Outside events, called *Triggers,* make the FSM evaluate its transition's guards and enable a transition to occur. In *Figure 1*, the system will transition from **Starting_Motion** to **Turning** once the robot motion is confirmed, or from **Turning** to **Stopping_Motion** if the compass is functioning correctly and its readings confirms the robot has turned by the specified angle. In this particular FSM, no specific Triggers have been defined, so any Trigger (such as a periodic "CLOCK" event) will make the FSM evaluate its transitions.

A set of states connected together by transitions forms a state *machine*. In *Figure 1*, the block **Turn_on_Spot_by_Angle** is an FSM that implements a behaviour that has a mobile robot turn on the spot. The FSM also contains parameters it uses to make decisions (such as the current robot heading and the commanded angle of the turn) or on which it acts (the robot itself).

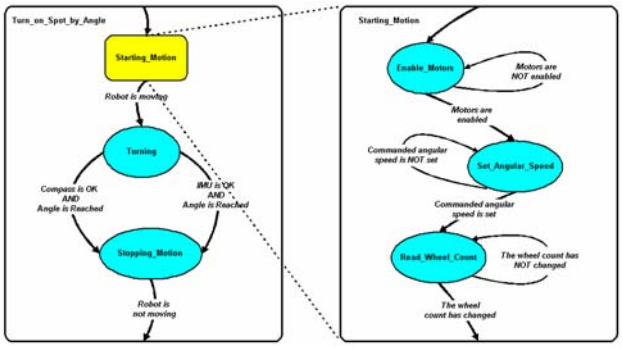

Figure 2 - A Hierarchical Finite State Machine

In *Figure 1*, **Starting_Motion** is represented as a single state. However the logic to be used in this state could be complex. In order to represent the state logic, the user can decide to use a sub-FSM in its place. The result is a *Hierarchical Finite State Machine* (HFSM) as shown in *Figure 2*.

In this case, the **Starting_Motion** sub-FSM can be made modular by specifying the robot on which it acts as an input parameter. Defining parameters required or produced by a FSM defines its interface to the outside world thus allowing its reuse in various higher levels FSMs, an approach that has been successfully used in software libraries for years. This is the strategy used by *Cortex* to provide FSM modularity and reusability.

3.1 Cortex Architecture

The architecture provides modules that implement the functionalities of the Cortex framework. *Figure 3* shows the two parts of the Cortex architecture. The *Development Environment* refers to the environment used by the developers to create, design, generate, deploy, control, command and monitor autonomy engines. The *Target System Environment* refers to the system where the autonomy engine is running. The modules are portable and have been developed with Java. They have been tested on Windows, Linux and Solaris. In addition, ARGO provides Java packages to interface with non-java code (e.g. Simulink, C, C++).

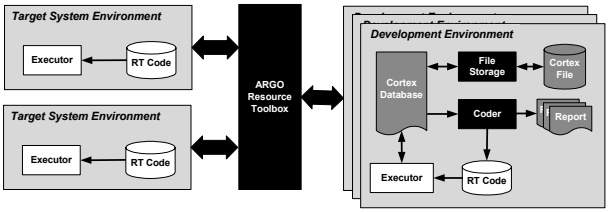

Figure 3 - Cortex Architecture

3.2 Graphical User Interface

Cortex modules are all bound into a single Graphical User Interface (GUI). This interface provides all the features required to execute all the steps mentioned in the previous section: It provides panels to

- edit and create Cortex projects;
- generate and compile real-time code;
- deploy the real-time code onto the target system;
- execute, command, control and monitor local and remote instances of the real-time code;
- playback previous state and parameter changes;

The use of an intuitive graphical representation of FSM (see *Figure 4* and *Figure 5*) by *Cortex* allows the developer/operator to concentrate on the problem to be

solved instead of concentrating on the programming skills to implement the solution.

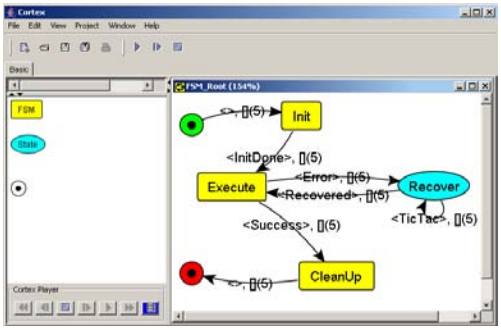

Figure 4 – Cortex GUI: FSM Editing

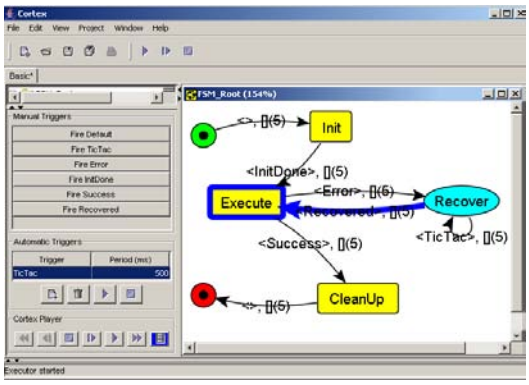

Figure 5 – Cortex GUI: FSM Monitoring

FSM are assembled graphically using States, sub-FSM, junctions (a construct used to combine transitions) and transitions. The operator can provide JAVA code snippets for state actions and transition's guard expressions and can define the inputs, local variables, and output parameters of each sub-FSM. The user can also assign a priority to each transition to force the order in which their guards are tested during execution. Constructs from other FSM can also be graphically "cut-and-pasted" to the current FSM to allow for the reuse of existing FSMs.

3.3 Cortex Code Generation

The current implementation of *Cortex* provides an automatic code generator that produces JAVA source code to implement the FSM defined by the user. The code generator provides FSM topology checking to detect unreachable states and transition loops. A compiler module then compiles the code, report problems, and helps the operator localise errors by highlighting FSM components where code is in errors.

The *Cortex Coder* generates real-time code mainly composed of conditional logic statements. It is free of threads and performs computation only on trigger invocations. It is self-contained and does not require the Cortex framework to be executed. The developer may decide to take the real-time code and integrate it by hand in his application. The *Cortex Coder* currently supports Java but its architecture will eventually support C++.

Cortex supports the distribution of autonomy among multiple systems: for example across several robots or throughout a distributed ground control station. A state from a Cortex project may submit a trigger to another Cortex project using the *ARGO REMOTE Toolbox*. It is also possible for multiple operators to control, command and monitor the same autonomy engine.

4 SAMPLE CASE

The *ARGO* framework has been applied to a few reference cases typical of space robotics applications. The application described in this paper is a satellite servicing application in Low-Earth orbit. This is representative of most robotic manipulation tasks in Earth orbit where the environment is known and structured but it is dynamic since the satellite to be captured is in free flight. Bandwidth limitations and communication dropouts dominate the quality of the communication link.

The sample application described below is a laboratory implementation of an autonomous satellite capture scenario based using an active vision system. This implementation is performed on the Canadian Space Agency's (CSA) Automation and Robotics Test-bed (CART) to validate the Cortex Toolbox in preparation for the TECSAS mission [11].

Figure 6- CSA Automation Robotics Testbed

In preparation for TECSAS, the *ARGO* technologies are being validated in laboratory on the CSA Automation and Robotics Test-bed (CART). This test-bed, shown in Figure 6, is composed of two 7-degrees-of-freedom manipulators. One of the manipulator arms is used to emulate the motion of the client satellite whereas the other emulates the motion of the manipulator on the servicer satellite.

The overall implementation of this OOS demonstration on the CART test-bed is presented in Figure 7. The overall control architecture of the two robotic arms is implemented in Matlab/Simulink. The execution code is automatically generated using the Real-Time Workshop toolbox of Matlab and is compiled and run on a cluster of Pentium IV computers operating under the realtime QNX environment.

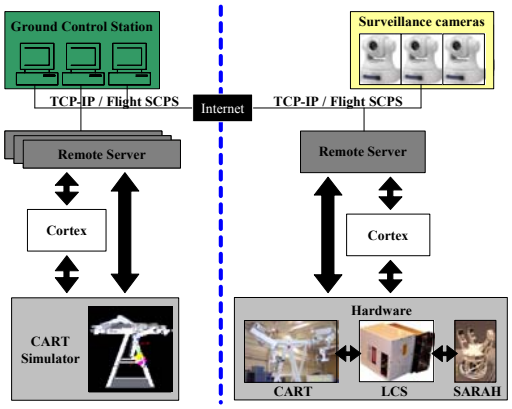

Figure 7: Overall implementation of the CART testbed

The *Cortex Toolbox* is used to implement the behaviours required for the autonomous capture of the client satellite. On TECSAS, the operator will be responsible for the planning and execution of the long-range rendezvous of the two spacecrafts. The autonomy engine will take control when the two spacecraft are distant by a few meters. It will be responsible for performing the final approach of the servicer spacecraft to the client, deploying the manipulator arm and performing the capture of the slow spinning/tumbling client satellite.

Transitions between phases of the operation are triggered by sensory events. The *Cortex* engine considers anomalies such as the possibility of the client spacecraft to drift out of the capture envelope of the manipulator (through translation or rotation), blinding of the vision sensor or loss of sight, reduction of the safe distance between the two manipulators below an acceptable limit, or failed capture which results in the client satellite to be sent into a tumble mode. *Figure 8* presents a *Cortex* implementation of a typical OOS scenario that would include an autonomous far rendezvous.

5 CONCLUSION

The Canadian Space Agency has developed the Autonomous Robotics and Ground Operations (*ARGO*) Framework for space robotic operations. The two objectives of *ARGO* are:

- To streamline the operation cycle by providing an integrated environment for planning, verification, execution and post-flight analysis.
- To reduce operations costs by enhancing the local decision-making capabilities of space robots through the inclusion of local autonomy.

One of the central building blocks of *ARGO* is the *Cortex Toolbox*. This toolbox is used to implement autonomy using the concept of hierarchical finite state machines. The fact that *ARGO* treats autonomous behaviours in the same manner as command scripts allows the operator to program, verify and uplink autonomous behaviours in the same development environment that is used for operations planning. Thus, *ARGO* truly provides an integrated environment for all operations-related issues from design and testing of autonomous behaviours to planning, verification and execution of command scripts.

A sample application of the *Cortex Toolbox* to a laboratory demonstration of a satellite-servicing mission is described. This demonstration is performed in preparation for the validation of the *ARGO* tools for their usage in the TECSAS satellite servicing technology demonstration mission.

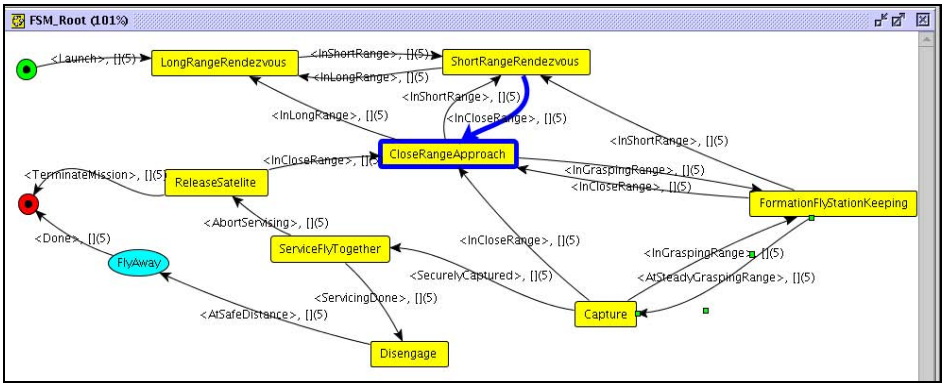

Figure 8: Cortex implementation of a typical autonomous On-Orbit Servicing scenario

6 REFERENCES

[1] L. Pedersen, R. Sargent, M. Bualat, C. Kunz, S. Lee, and A. Wright, "Single-Cycle Instrument Deployment for Mars Rovers", *Proc. of 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Nara, Japan, May 2003.

[2] http://www.mdrobotics.ca/iiro.htm

[3] E. Dupuis, R. Gillett, P. Boulanger, E. Edwards and M. Lipsett, "Interactive, Intelligent Remote Operations: Application to Space Robotics", SPIE Telemanipulator and Telepresence Technologies VI, Boston, Septembre 1999.

[4] E. Dupuis and R. Gillett, "Remote Operation with Supervised Autonomy", 7th ESA Workshop on Avanced Space Technologies in Robotics and Automation (ASTRA 2002), Noordwijk, The Netherlands, November 2002.

[5] Background, challenges, and solutions for guidance, navigation, and control for satellite servicing, *DARPA Presentation on the Industry Day for Orbital Express Program*, Nov. 1999.

[6] B. Brunner, K. Landzettel, G. Schreiber, B. M. Steinmetz and G. Hirzinger, "A Universal Task-Level Ground Control and Programming System for Space Robot Applications - the MARCO Concept and its Applications to the ETS-VII Project", Proc. Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS 99), ESTEC, Noordwijk, The Netherlands, pp.217-224, June 1-3 1999.

[7] T. Huntsberger, P. Pirjanian, A. Trebi-Ollennu, H. Das, H. Nayar, H. Aghazarian, A. J. Ganino, M. Garrett, S.S. Joshi and P. S. Schenker, CAMPOUT: a control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration IEEE Transactions on Systems, Man and Cybernetics, Part A, Volume: 33 , Issue: 5 , Sept. 2003 Pages: 550 – 559

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

[9] I.A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, "CLARAty and Challenges of Developing Interoperable Robotic Software," invited to *International Conference on Intelligent Robots and Systems (IROS),* Nevada, October 2003.

[10] R. L'Archevêque and E. Dupuis, "Autonomous Robotics and Ground Operations", Proc. of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space – iSAIRAS 2003, Nara, Japan, May 2003.

[11] B. Sommer, On-Orbit Servicing of Satellites (OOS) as a major application field – The TECSAS mission, 54th International Astronautical Congress of the International Astronautical Federation (IAF); Bremen, Germany; Sep. 29 - Oct. 3, 2003.