CASE: A COGNITIVE ARCHITECTURE FOR SPACE EXPLORATION

*Pete Bonasso[1], Stephen Hart [2], Ana Huaman[3], James Kramer[4]

[1]TRACLabs, Inc, 16969 N. Texas Ave, Suite 300, Webster, TX 77059, USA, E-mail: bonasso@traclabs.com

[2] TRACLabs, Inc, 16969 N. Texas Ave, Suite 300, Webster, TX 77059, USA, E-mail: swhart@traclabs.com

[3]TRACLabs, Inc, 16969 N. Texas Ave, Suite 300, Webster, TX 77059, USA, E-mail: ana@traclabs.com

[4] TRACLabs, Inc, 16969 N. Texas Ave, Suite 300, Webster, TX 77059, USA, E-mail: jkramer3@traclabs.com

ABSTRACT

Because close collaboration between the crew and mission control will not be practical for inter-planetary exploration, NASA envisions the need for an intelligent autonomous agent that can continually integrate data from the spacecraft or lunar/planetary base to advise the crew during their mission. Over the past several years, TRACLabs, in support of NASA and other government agencies, has developed a number of software components that can be used in such an agent, and has developed an architecture called a cognitive architecture for space exploration (CASE). This paper describes that architecture and discusses its feasibility for use in space exploration.

1 INTRODUCTION

In space operations, carrying out the activities of mission plans by executing procedures often requires close collaboration between ground controllers who have deep knowledge of the spacecraft's systems and crewmembers who have in-situ situation awareness. Because of the light

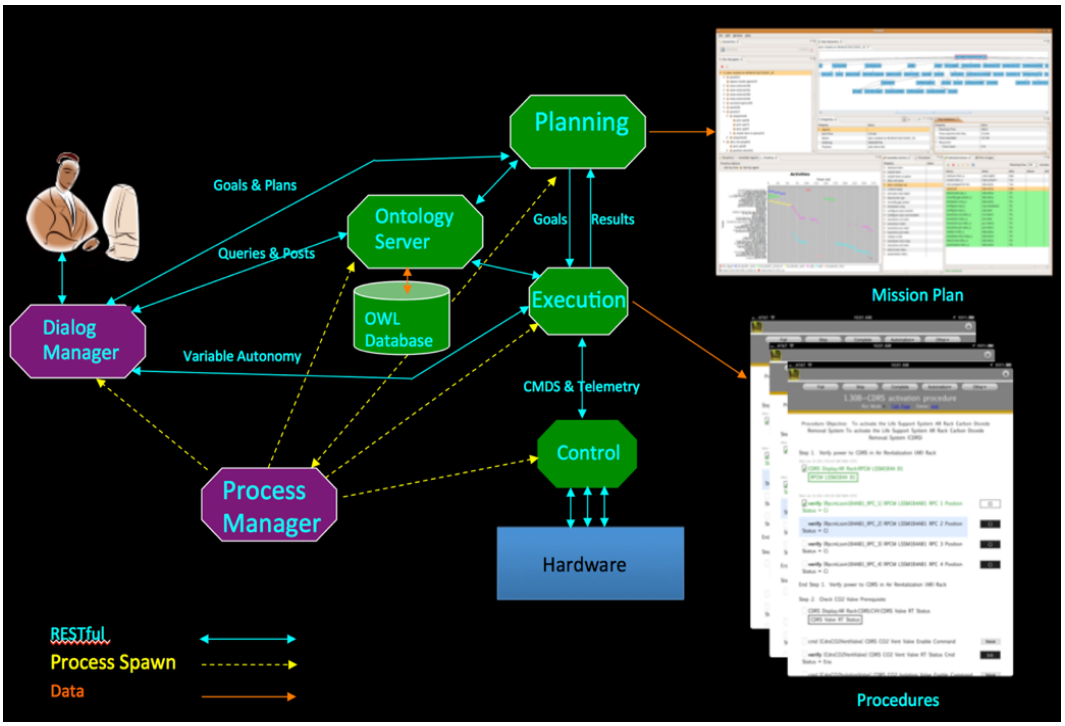

Figure 1 The top-level CASE architectural design. The planner generates an activity plan whose primitives are procedures run by the execution system, which in turn interfaces to the hardware via control software. The planning and execution systems draw information from and update the system ontology, stored in the Web Ontology Language (OWL) format. A process manager spawns the main components of the system, and re-spawns processes if their supporting computing infrastructure fails. The user interacts with the agent via a dialog manager, which has access to all of the running processes.

distances involved, this close collaboration will not be practical for inter-planetary exploration. In this work, we are developing a software cognitive architecture for space exploration (CASE) that will autonomously carry out exploration operations by using the same knowledge and executing the same plans and procedures as those developed on Earth.

The arrangement of software components and their internal interfaces is called an architecture, and if those components individually or together are capable of cognition – awareness through perception, and capable of acquiring knowledge, solving problems and communicating via language – it is called a cognitive architecture. Over the past several years, TRACLabs, in support of NASA and other government agencies, has developed a number of software components that can be used in such an architecture, and is now designing an exploration agent based on that architecture and showing that it is feasible for use in space exploration. These components include a procedure development and execution system known as PRIDE [1] that allows for variably autonomous execution of both crew and robotic procedures, an automated planner [2] that plans and re-plans the execution of procedures to achieve overall mission goals and an ontology data management system [3] that makes consistent system states available to all the components. Since there will be rovers and other robots in a planetary base, the architecture also supports supervised robotic activities as well.

Also, we contend that any useful cognitive architecture must include components to support human monitoring and human-machine interaction, particularly for robot systems on a planetary surface. So, our proposed architecture includes not only monitoring and control components, but also a natural language (NL) dialog system, and a variable autonomy procedure execution system. Finally, to be robust, the architecture should be modular, and be capable of supporting multiple processes executing on multiple processors, and it should be robust to processor failure.

So, besides integrating the existing components into a coherent reasoning architecture, we have added an NL dialog system and a process manager to track running processes and restore them when they fail (see Figure 1). The remainder of this paper describes our simulation environment, the components of the architecture in more detail, a demonstration of the working system, and discusses desired future work to improve CASE.

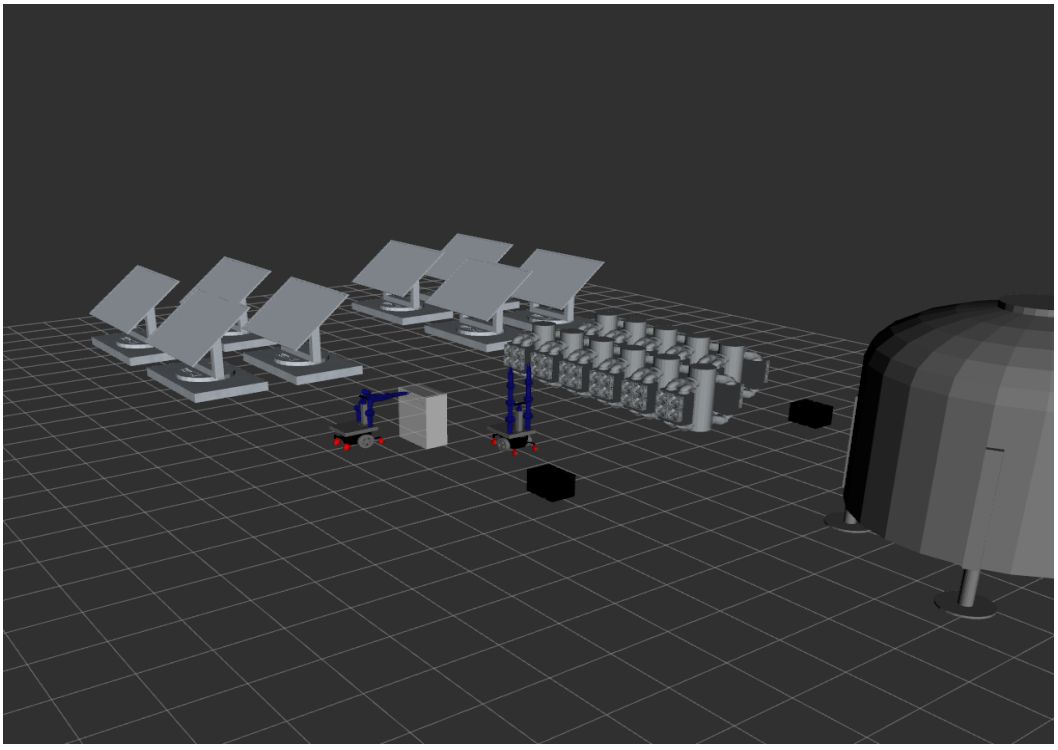

Figure 2 The planetary base. The solar panels are in the background, the batteries are in the right mid-ground, and the habitat is in the right foreground. One two-armed robot stands at the switching panel; a second stands near one of two black DC to DC Converter Units (DDCUs).

2 A SIMULATION FOR CASE DEVELOPMENT

We have selected a planetary base scenario as depicted in Figure 2. The power system consists of batteries, solar panels, DC to DC Converter Units (DDCUs), that provide direct power from the batteries to the habitat, and a control panel with switches for connecting solar panels to batteries, batteries to DDCUs and DDCUs to the habitat. There is a crew of six, each of whom is qualified to fulfill roles for activities, such as power and thermal control, life support, robotics and extra-vehicular activity (EVA). Life support systems modeled in the habitat included the internal active thermal control system (IATCS), the carbon dioxide removal system (CDRS), and the oxygen generation system (OGS).

To simulate our planetary base, we are using a test bed developed for a National Space Biomedical Research Institute (NSBRI) Supervisory Control Study [4], which includes a habitat with systems powered from batteries charged from solar panels, a humanoid robot that connects batteries to solar panels or to power distribution grids for the habitat and a life support simulation called BIOSIM [5]. The activities we have modeled for planning and execution include connecting various electrical power systems, activating and de-activating the IATCS, CDRS and OGS, preparing and donning spacesuits, charging the rovers and driving or supervising their traverses. The above activities can be planned by our automated planner, AP [2], and the low level tasks, such as connecting a solar panel to a battery or activating the IATCS are executed by procedures developed via the PRIDE system. Also, while the making of electrical connections are overseen by a crewmember, they are carried out by a two-armed robot, and if the robot has some trouble grasping and turning a switch on the control panel, the crewmember can adjust its grasp via what we call an affordance template (AT) [6] (see Section 7 below).

3 PLANNING AND EXECUTION

The backbone of CASE is the planning and execution system shown in Figure 1. This system is an instance of layered software control [7]. Our automated planning system generates plans whose primitive actions (executables) are procedures developed in our PRIDE system. A PRIDE executive called the procedure agent for execution, or PAX, runs the procedures, which are represented in the procedure representation language (PRL) [8]. The procedures can be fully automated or a mixture of automated and manual instructions, executed by both a crewmember and the system. PAX interfaces with the hardware via flight software, be it vehicle, process control or robotic. As the control software executes, PAX updates the system states in the ontology and informs the planner of the success or failure of the plan step. After each update the planner may re-plan to accommodate failures or unexpected successes.

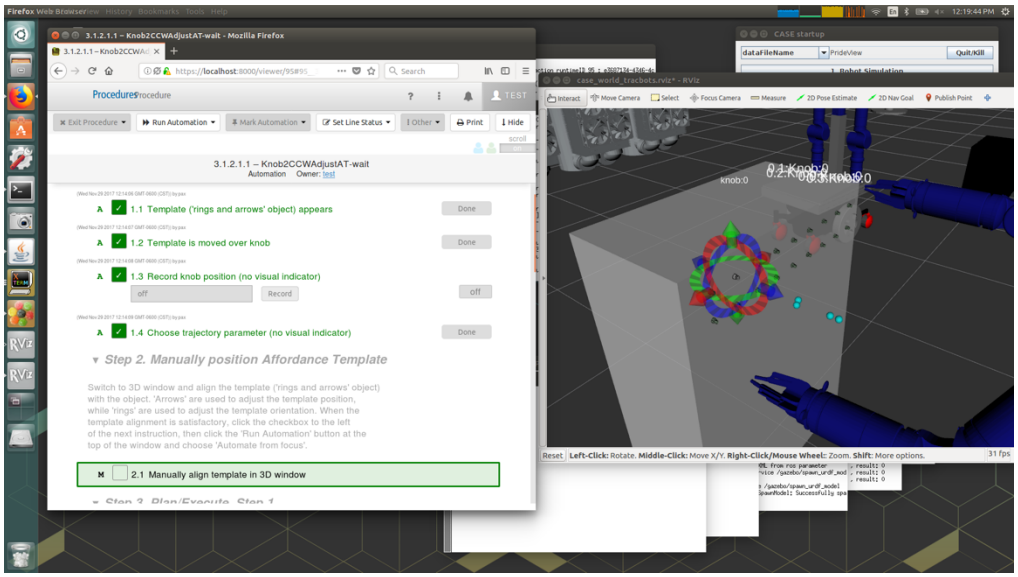

Figure 3 A procedure to allow a user to adjust the affordance template (AT) for a knob on the control panel. The procedure has paused on a manual step to allow the user to adjust the template.

4 ONTOLOGY SERVER

To review, an ontology is a rigorous, exhaustive organization of the knowledge of a domain, containing all relevant entities and their relations. We used the Web Ontology Language (OWL) [9] to construct models of the International Space Station (ISS) to be used during the generation of activity plans. Our OWL models consist of classes, instances and relations among them, such as (is-attached-to DDCU3 Coldplate1); axioms, for example, if a container is moved then all of its contents move as well; and of course, the information – data and object properties – associated with each entity. The users can modify the ontology via the PRIDE Ontology Editor (PRONTOE) [10]. In this project, the ontology serves as the long-term memory of CASE. It consists of the ISS ontology, extended to include the systems and subsystems for a planetary base and its associated equipment such as rovers.

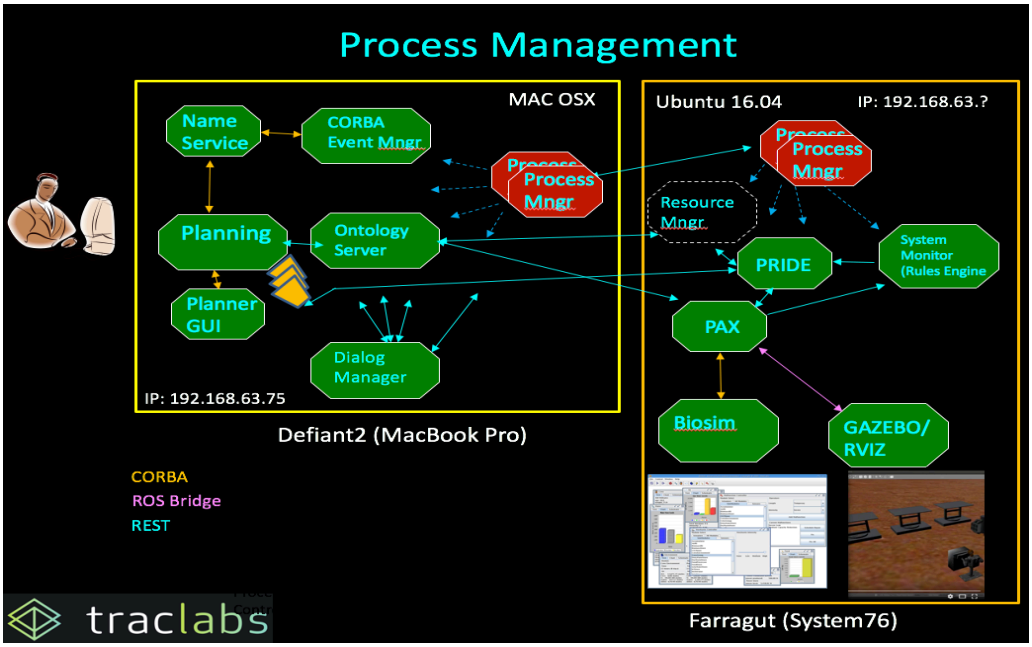

Figure 4 The CASE processes and their distribution across a MacBook Pro and a high-speed Linux workstation. The yellow polygons represent executive bridges that translate the planning actions into procedure invocations. The resource manager is an external process discussed in the conclusions. Our communications configuration includes CORBA [11], RESTful [12] and a robot operating system (ROS)[13] bridge between PAX and the robot simulations.

5 DIALOG MANAGER

The previously developed components have interfaces, such as the interactive GUI for the planner, a procedure execution viewer, and the rendering of the planetary base (see Figure 2). But we also developed a dialog manager, which allows the crew to interact with the agent using natural language.

The dialog manager has three functions: to fulfill user information requests, to serve as a command interface to the CASE agent procedures, and to post new goals or delete obsolete goals from the activity plan. Our dialog system is based on the Dynamic Predictive Memory Architecture (DPMA) [14], which arose from three basic ideas. The first is the use of direct memory access parsing (DMAP) [15] to process a sentence in natural language; the second is that natural language processing is less complex when the corpus of semantic concepts can be circumscribed by a particular domain using an ontology; and the third is the view that dialog is a planful activity [16]. Rather than syntactic and semantic parsing of natural language, DPMA processes the utterance text by matching phrasal patterns attached to nodes in a concept memory, and we generate CASE's concept graph directly from the domain ontology. So, the sentence, "Where is Rover-1?" is matched to a Request For Location Information concept, which has placeholders for a conceptual object, which in this case is the mobile robot, Rover-1.

When the cognitive architecture and the user share a common set of beliefs about the applications domain and how to get things done, it is relatively easy to determine what words in the utterance are ambiguous. For example, in the query, "Where is the rover?" the conceptual object is a class of vehicle but is not specific enough to execute the query. Given that the dialog has the overarching

purpose to satisfy the user's informational goals, our query processing system is a turn-based planning function, wherein the dialog continues until the user's needs are satisfied. For example, the system might respond, "I understand you are requesting location information about a rover. There are two rovers serving our base. To which are you referring?" and present a list of active rovers from which to choose.

6 PROCESS MANAGER

It seems reasonable to assume that the computing resources used to host CASE could become compromised, and thus, require some mechanism to reconstitute those processes and continue operations. Our basic approach is to have a process manager (PM), along with a redundant copy, running on each machine that can host processes. A parent process orchestrates the starting, stopping and restarting of processes on its machine, or, via RESTful commands, uses sibling processes on other machines. Process managers monitor their processes and, should one fail, reconstitutes that process as well as all processes that depend upon it.

The key to having our process manager be more than a simple built-in recovery mechanism, such as Linux's systemd, is to maintain knowledge of the dependencies each process has with the others. For example, if in running CASE we lose PRIDEView, we need to reconstitute PAX as well as the PAX processes for each executing agent. So, the process manager(s) maintain a table of dependencies, to insure all dependent processes are stopped/restarted in their order of mutual support.

7 ALLOWING THE USER DIRECT CONTROL

As with all autonomous systems, the human user wants to have access to any part of the system, if for nothing else than to see details of the operation. In CASE, procedures can be authored to allow low-level access to the robots in the simulation (they can also be written to access any low-level command or telemetry for the life support system). In Figure 3, we see a procedure that has a manual instruction to adjust the affordance template (AT) of the robot for a particular knob on the control panel. The user has made visible the AT involved in the task and can adjust any of the six degrees of freedom for the grasp. When the user has

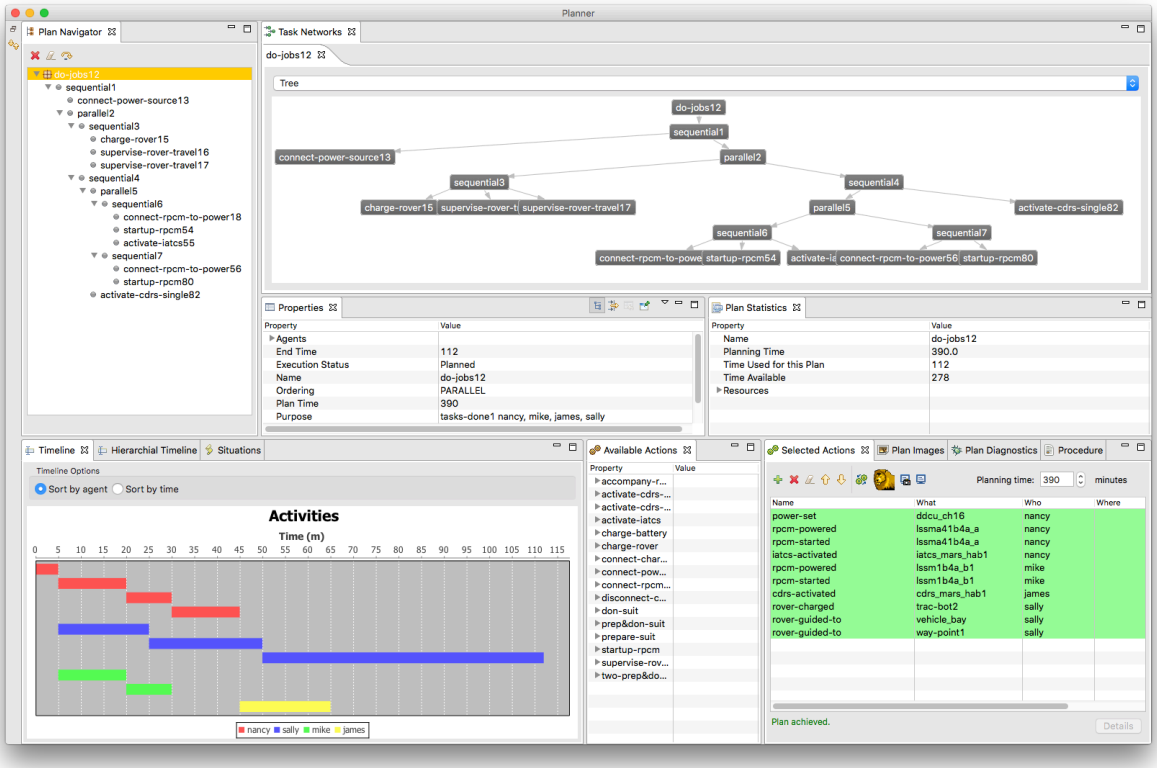

Figure 5 A view of our interactive planning interface and the demonstration plan.

completed the adjustment, she marks the manual instruction as complete, and the automation saves the new AT and continues the procedure until the grasp is complete.

8 A CASE DEMONSTRATION

We have developed a CASE demonstration capability that uses a MacBook Pro as the primary computer and a high-end Linux computer as the secondary computer. The planning and execution processes, ontology server and dialog manager run on the Mac, while the PRIDE processes and the simulations run on the Linux machine (Figure 4). This configuration is preferred, since we want the PRIDE software running on the same machine as the simulations if at all possible, while the higher levels of the architecture run on the primary computer.

8.1 The Main Demonstration

At the planner GUI (Figure 5) the user selects the tasks to be done and assigns agents (selected actions pane in Figure 5). She then clicks on the execute icon (the computer icon in the selected actions pane) to generate the demonstration plan shown as a hierarchical task net (top right of Figure 5), an indented plan navigator display (top left of Figure 5) and as an activity timeline (bottom left of Figure 5). In the resulting plan, the first task will provide power to a habitat DDCU. The next three tasks power and then guide a rover to two locations. The next three tasks close switches so that power will flow to the IATCS and then activate the IATCS. The last three tasks carry out a similar function to provide power for and to activate the CDRS.

In the timeline of Figure 5, Nancy (red activities) must power the DDCU and then execute the other three tasks concerning getting the IATCS started. In parallel, Sally (blue activities) charges the rover and supervises its travel. Also, in parallel, but after the DDCU is connected to the habitat, Mike (green activities) is assigned to power the CDRS, but only James (yellow activity) is qualified to activate the CDRS.

The user then selects the plan in the navigation display and clicks on the execute icon (half circle arrow in the plan navigator pane in Figure 5). From that point on, though the user must take some manual actions in the procedures, the system runs autonomously to completion. As each task becomes ready to execute, the planner sends the task to the assigned agent's executive bridge (see Figure 4), which starts up the appropriate procedure in PRIDE. When PAX indicates the procedure is complete, the bridge notifies the planner, which then updates the plan, advances the time line and sends the next set of ready tasks.

8.2 Process Management

We can currently demonstrate many combinations of recoveries from failed processes, but the main ones we considered concerned loss of some process in each of the communications protocols (see Figure 4), as well as loss of the PMs themselves. So, the following can be demonstrated:

a) Loss of a redundant PM: Artificially kill the redundant primary or sibling PM; the primary or sibling PM restarts the redundant PM.

b) Loss of a primary PM: On either machine, artificially kill the primary or sibling PM; the redundant primary or sibling PM restarts the redundant PM.

c) Loss of the ontology server: Artificially kill the ontology server; the PMs first shutdown the planning GUI, the exec-bridges and AP, and then restart the ontology server and then AP, the exec-bridges and the planning GUI.

d) Loss of PRIDEView: Artificially kill the PRIDEView process; the PMs first shutdown PAX and the exec bridges, then restart PRIDEView, PAX and the exec bridges.

e) Loss of the CORBA Name Service (used by the planner and its GUI): Artificially kill the CORBA Name Service; the PMs first shutdown the planner GUI, exec bridges, AP, and the CORBA General Factory, then restart the name service, and then the CORBA Factory, AP, exec bridges and the planner GUI.

As a variation, we can cause the redundant PM on the primary machine to temporarily halt its monitor loop then shutdown the primary PM and then the ontology server. When we restart the monitor loop in the redundant PM, the redundant PM immediately restarts the parent PM, which then carries out the actions in c) above.

9 CONCLUSIONS AND FUTURE WORK

We believe CASE will provide a feasible approach to agent design for space exploration, provide on-board autonomy in nominal operations and human-computer solutions for off-nominal operations and be robust in the face of computational failures.

Our future work with CASE involves more extensive process management, consolidating the disparate user interfaces and developing a canonical interface for external modules.

6.1 Improved Process Management

Section 8.2 described the types of failure scenarios we have investigated. While CASE will reconstitute all the appropriate processes, only PRIDE has a record of what had occurred prior to the failure. As a result, the demonstration can restart "from the top" and so is still functional. However, in an actual planetary situation, trying to activate the life support systems when they're already started will cause the activation procedures to fail, which of course will cause the planner to fail the plan, without recourse.

The main problem is that there is no check-pointing going on. The planner is fully capable of re-planning from where it left off, if only it could save the situation at that point in time. The ontology server is updated whenever a state changes in the world, but it needs to write out those states periodically and reload them when it recovers from failure. Developing check pointing throughout the system will be a future endeavor.

Finally, we want to include hardware requirements, e.g., memory, processor speed and bandwidth, for each process, so that the PMs can reconstitute processes on secondary machines if the original machines loose power.

6.2 A Canonical Interface for External Modules

An "external" module is one that is not part of CASE proper, such as applications developed by NASA international partners. An example is the resource manager (RM) in Figure 4. The RM monitors procedures and provides warnings when resources are nearly depleted. In order to accommodate external processes, we are developing a canonical interface based on the Agent Communications Language (ACL) (http://www.fipa.org/repository/aclspecs.html) and the semantics developed for the Knowledge Query and Manipulation Language [17]. These can be distilled into ontological references and "performatives" given in semi-natural language speech acts. The point of these performatives is to abstract the details of the actual RESTful queries and posts so that the authors of the external processes need not deal with them.

Since all processes in CASE use the same ontology, we need not include an ontology reference in our performatives, which will include a verb, a subject, an object and an information-object. For example, a query for a list of procedures would contain the verb "ask", the subject "PRIDE" and the information, "available procedures". A query for the attributes of an object in the ontology would be the verb "ask", the subject "ontology" and the object, e.g., "rover1". When the RM wants to post information to a process, say, PAX, the verb would be "send comment", the subject, "PAX", and the information-object, "running procedure 31 is using camera3".

6.3 The FACE of CASE

Past and current work in NASA space analogs, such as NEEMO (https://www.nasa.gov/mission_pages/NEEMO/index.html) and HERA (https://www.nasa.gov/analogs/hera) has shown that the usefulness of a cognitive architecture, or indeed, of any automation supporting human activity, is only as good as its user interfaces and how well they support effective human-automation interaction and collaboration. The agent should include user-centered design [18] of its displays and interaction modes to help users maintain situation awareness [19], and should recommend courses of action without undue cognitive load or distraction, particularly for the intra-vehicular activity user, or IVA. The users should be able to easily and quickly be brought up to speed about anomalous situations so that they can deal with the problem while maintaining the status quo of nominal operations. We are in the process of designing a canonical interaction protocol for CASE, thus providing an exemplar for a flexible agent-based communication for exploration (FACE).

Acknowledgement

This work was funded under NASA contract NNX17CA59P.

References

[1] Izygon, M., D. Kortenkamp, and A. Molin, *A procedure integrated development environment for future spacecraft and habitats.*, in *Space Technology and Applications International Forum (STAIF)*. 2008, American Instutute of Physics: Albuquerque, NM.

[2] Elsaesser, C. and J. Sanborn, *An Architecture for Adversarial Planning.* IEEE Transactions on Systems, Man, and Cybernetics, 1990. **20**(1): p. 186-194.

[3] Bell, S., et al., *PRONTOE: A case study for developing ontologies for operations*, in *5th International Conference on Knowledge Engineering and Ontology Development (KEOD 13)*. 2013: Algarve, Portugal.

[4] Schreckenghost, D., D. Billman, and T. Milam, *Effectiveness of strategies for partial automation of electronic procedures during nasa hera analog missions*, in *International Joint Conferences on Artificial Intelligence (IJCAI) Workshop on AI and Space (IWPSS)*. 2015: Buenos Aires, Argentina.

[5] Kortenkamp, D. and S. Bell, *BioSim: An Integrated Simulation of an Advanced Life Support System for Intelligent Control Research*, in *The 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-03)*. 2003: Nara, Japan.

[6] Beeson, P., S. Hart, and S. Gee, *Cartesian motion*

planning & task programming with CRAFTSMAN., in *RSS 2016 Workshop on Task and Motion Planning*. 2016: Ann Arbor, Michigan.

[7] Gat, E., *Three-Layer Architectures*, in *Mobile Robots and Artificial Intelligence*, D. Kortenkamp, R.P. Bonasso, and R. Murphy, Editors. 1998, AAAI Press: Menlo Park, CA. p. 195-210.

[8] Kortenkamp, D., R.P. Bonasso, and D. Schreckenghost, *A Procedure Representation Language for Human Spaceflight Operations*, in *The 9th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-08)*. 2008: Los Angeles, CA.

[9] OWG. *the Web Ontology Language (OWL)*. 2004; Available from: http://www.w3.org/TR/owl-features/.

[10] Bell, S., et al., *PRONTOE: An Ontology Editor for Domain Experts.* Communications in Computer and Information Science, 2015. **454**: p. 153-167.

[11] OMG. *CORBA Basics*. 2002; Available from: http://www.omg.org/gettingstarted/corbafaq.htm.

[12] Fielding, R.T., *Architectural Styles and the Design of Network-based Software Architectures (chapter 5: Representational State Transfers)*, in *Computer Science*. 2000, UC Ervine: irvine, CA.

[13] Quigley, K., et al., *Ros: an open-source robot operating system*, in *ICRA Workshop on Open Source Software*. 2009: Kobe, Japan.

[14] Fitzgerald, W. and J.R. Firby. *The Dynamic Predictive Memory Architecture: Integrating Language with Task Execution*. in *Proceedings of IEEE Symposia on Intelligence and Systems*. 1998. Washington, DC.

[15] Martin, C.E., *Direct Memory Access Parsing. Ph.D. Dissertation* 1990, Yale University.

[16] Grosz, B.J. and C.L. Sidner, *Plans for Discourse*, in *Intentions in Communication*, P.R. Cohen, J. Morgan, and M.E. Pollack, Editors. 1990, MIT Press: Cambridge, MA. p. 417-444.

[17] Finin, T., Y. Labrou, and J. Mayfield, *KQML as an Agent Communication Language*, in *Software Agents*, J.M. Bradshaw, Editor. 1997, AAAI Press / The MIT Press: Cambridge, MA. p. 291-316.

[18] Norman, D., *The Design of Everyday Things: Revised and expanded edition*, ed. B.B. (AZ). 2013.

[19] Endsley, M.R., Automation and situation awareness. *Automation and human performance: Theory and applications*, 1996. **20**: p. 163-181.