NASA/TP—2012–216041

Lunar Surface Systems Software Architecture Study:

Open Architecture

William J. Clancey
Ames Research Center, California
and Florida Institute for Human and Machine Cognition, Pensacola

Robert Nado, Ron van Hoof, Mike Lowry
Ames Research Center

Grailing Jones and Daniel Dvorak
NASA Jet Propulsion Laboratory

August 2012

NASA STI Program ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 443-757-5803

- Phone the NASA STI Information Desk at 443-757-5802

- Write to:
  STI Information Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/TP—2012–216041

Lunar Surface Systems Software Architecture Study:

Open Architecture

William J. Clancey
Ames Research Center, California
and Florida Institute for Human and Machine Cognition, Pensacola

Robert Nado, Ron van Hoof, Mike Lowry
Ames Research Center

Grailing Jones and Daniel Dvorak
NASA Jet Propulsion Laboratory

Presented at
Space Mission Challenges for Information Technology Conference (SMC-IT)
IEEE
Palo Alto, CA, August 2011

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, CA 94035-1000

August 2012

Acknowledgments

This study was supported by the ETDP/Lunar Surface Systems Project in 2010. Funding for the Mobile Agents Project (2002-08) was provided by NASA's Intelligent Systems, Moon and Mars Analogue Mission Activities (MMAMA), and Exploration Technology and Research Programs (ETDP). Scientists and engineers from three NASA centers contributed to Mobile Agents, including Rick Alena, John Dowding, and the JSC Scout/ERA team (especially Rob Hirsh, Jeff Graham, and Kim Shillcutt Tyree); we acknowledge also the guidance of the geologists, Brent Garry and Abby Semple, who experimented with the system doing field science at MDRS and other analog sites.

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
443-757-5802

National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312
703-605-6000

This report is also available in electronic form at

http://ti.arc.nasa.gov/publications/

TABLE OF CONTENTS

FIGURES

1 Executive Summary

This report is part of an overarching Lunar Surface Systems (LSS) Software Architecture Trade Study that identifies candidate architectures for the key software that will be used for each LSS Element.[1] One goal is to estimate the level of effort and cost required for software development relative to architectural features and system capabilities.

The report systematically analyzes data from the Mobile Agents Project, funded by NASA's Intelligent Systems and Human-Systems Integration Exploration Technology Development Programs (2002-2006) with the objective of developing an open architecture for an end-to-end exploration system focusing on science EVAs in which the crew cannot rely on ground support.

The Mobile Agents Architecture (MAA) provides a common goal-oriented, model-based interface that automates communication of information and commands in a distributed, concurrent system of systems, consisting of a diversity of hardware and software systems interacting in a mobile, distributed environment. The objective of this report is to use the engineering data from the sequence of field experiments to recommend informative metrics for software open architecture. This report also provides the groundwork for a Phase 2 interoperability trade study using prescribed workflows in EVA scenarios (Clancey and Lowry, 2012).

In particular, this study sought data to access whether the incremental buildup of an exploration system for long-duration capabilities is facilitated by an open architecture whose APIs are specifically designed to facilitate integration of new components, and thus minimize costs by reducing changes to the existing system and consequent rework. The study shows the advantages of *composite APIs* that map conventional component APIs (which provide access to the functional methods and data objects of components) to a service-oriented language through which people and subsystems (including external interfaces) communicate. For the Mobile Agents project, this service-oriented language is in terms of *messages about tasks* (e.g., an EVA plan, astronauts, life support, tools such as cameras, robotic assistants) and the external interfaces include voice commanding. This kind of composite API effectively enables a diversity of hardware and software components (including COTS) to provide *task-oriented services* to the overall exploration system.

In the mature software architecture described here, called the *Mobile Agent Architecture* (MAA) an open architecture consisting of *a workflow backbone of services ("agents") and composite APIs enabled distributed, mobile agents to handle simultaneous goal-oriented requests on a non-reliable network.* This architecture was developed from field experience with different existing hardware and software systems for both field science and routine habitat operations. The key architectural lessons concern how distributed mobile subsystems communicate, how task-level information and commands are

[1] See Clancey et al. (2011) for a conference paper version of this report.

represented, how services operating on the surface communicate with remote support teams, and how asynchronous services manage multiple, simultaneous requests.

This report also succinctly considers a related open-architecture developed at JPL that is centered on *goals* as the external interface language and services. Goals are constraints on state variables that are monitored and actively maintained. The qualitative experience is that a goal-oriented architecture is more robust than conventional control software, and more conducive to adding new components (see Appendix V).

1.1 Field Experiments and Data Analyzed

The field experiments for MAA were based on the methodology of *empirical requirements analysis*, in which prototype exploration systems were used for assisting crew members in simulated surface missions. In particular, the project explored how existing components (robots, cameras, computers, biosensors, GPS devices, electric power systems, databases, email, heads-up display, etc.) could be made into an integrated exploration system that was easily reconfigured for different EVAs and settings.

The data analyzed in this report consists of four different kinds of configurations or "product lines" comprising ten systems designed, developed, and tested by NASA Ames and JSC from 2002 through 2008:

- *"Automating Capcom" Configurations:*
    - **DRATS02**, **MDRS03**, **MDRS04** (all using EVA robotic assistant)
    - **MDRS05**, **DRATS05** (Scout vehicle), **CDS05**(K9 & Gromit robots)
    - **DRATS06** (pressurized suits, JSC ExPOC, and GeoPhone Array)
- *"Power Agents" Configuration:* **MDRS06**
- *Metabolic Advisor Configuration:* **POGO07**
- *iMAS Scientist's Field Assistant:* **MMAMA08** (HI, NM, Belize)

Exploration system configurations can be viewed *structurally* in terms of hardware and software components (e.g., a planning system) and integrative software (agents), and *functionally* in terms of workflow "capabilities" provided to the astronaut crew. A *workflow capability*, as the name implies, pertains to the flow of information requests, commands, and work products, initiated by either people or software in the context of a crew's work activity. Workflow capabilities usually take the form of requests for information that require data to be interpreted (e.g., how many hours will the batteries on some device last given current draw and planned usage?) or operations for subsystems to perform that require automated coordination of subsystems over time (e.g., "K9, inspect the area around waypoint 5"). Workflow capabilities also include direct requests for data readouts ("what is the current battery voltage?") and primitive subsystem operations ("Scout, turn on headlights"). Some functions require ongoing monitoring of sensor data ("Tell me when the generator is off-line").

Overall 134 workflow capabilities were developed for astronaut health monitoring, system health monitoring, location tracking, human-robot coordination, plan

management, science data logging, voice command interface controls, and alert management. The Mobile Agents systems were developed to illustrate typical functionalities that may be useful to scientist-astronauts during EVAs in which real-time communication with mission support is not possible due to time delay (Clancey 2004a; b). Voice commanding capabilities were designed to assist creating documented EVA products while flexibly following an EVA plan, keeping on route and schedule, and remaining aware of logistic/safety constraints and limitations.

The sources of data analyzed include: code repositories; project reports, schedules, email, and budgets; and expedition records. Analysis produced statistics about *software modification* (e.g., direct reuse or number of lines of code added) and *personnel effort* (e.g., size and distribution of teams; FTE for agents and integration only, not subsystems). Statistics were charted to determine advantages of the open architecture for phased development of LSS capabilities, such as adding robotic systems in the same production line (e.g., MDRS05 added a second ERA to MDRS04), adding a new robotic system by adapting existing software (e.g., Scout), and incorporating existing capabilities for different purposes (e.g., creating PA06 from DRATS05). Productivity calculations reflect the different kinds of work required for revising the system in these ways. In particular, a distinction is made between: *Reusing* a task-level service (no change), *adding functionality* to a service, and *adapting* a service for a different subsystem.

Analysis also examined the ability to reconfigure the exploration system for different work contexts, such as *adding new kinds of external systems* (shifting among science instruments, electric power systems, and life support systems) and *directly using existing services (e.g., email alerts) for new purposes or changing communication media* (e.g., from email to HUD to voice loop) without modifying code. Cost for these changes was estimated and correlated by counting workflow capabilities, thousands of source lines of code (KSLOC), and programming time comparatively in a series of system reconfigurations.

1.2 Analysis Overview

The analysis confirmed three primary hypotheses:

1. *An agent-oriented workflow system using composite APIs enables integrating components (e.g., biomedical algorithms, robots, databases) without modifying them.*
2. *Cost/capability is not increasing as new capabilities are added.*
3. *KSLOC/capability is not increasing as new capabilities are added.*

Analysis showed that a *workflow service backbone* was established in the third year (MDRS05) and was reused for subsequent configurations, ranging from 85% carryover when entirely different robots and a planning system were incorporated to 100% carryover when shifting from an EVA system to a habitat power monitoring system. At the same time, in developing the workflow service backbone, *nearly half of the APIs were unchanged.* Architecture changes focused on reconfiguring workflow services to respond to interface requirements, rather than re-integration with subsystems. That is, the

data show that *enhancement of workflow capabilities is often possible without modifying existing systems and their APIs,* in an open architecture in which integration occurs through task-level communications.

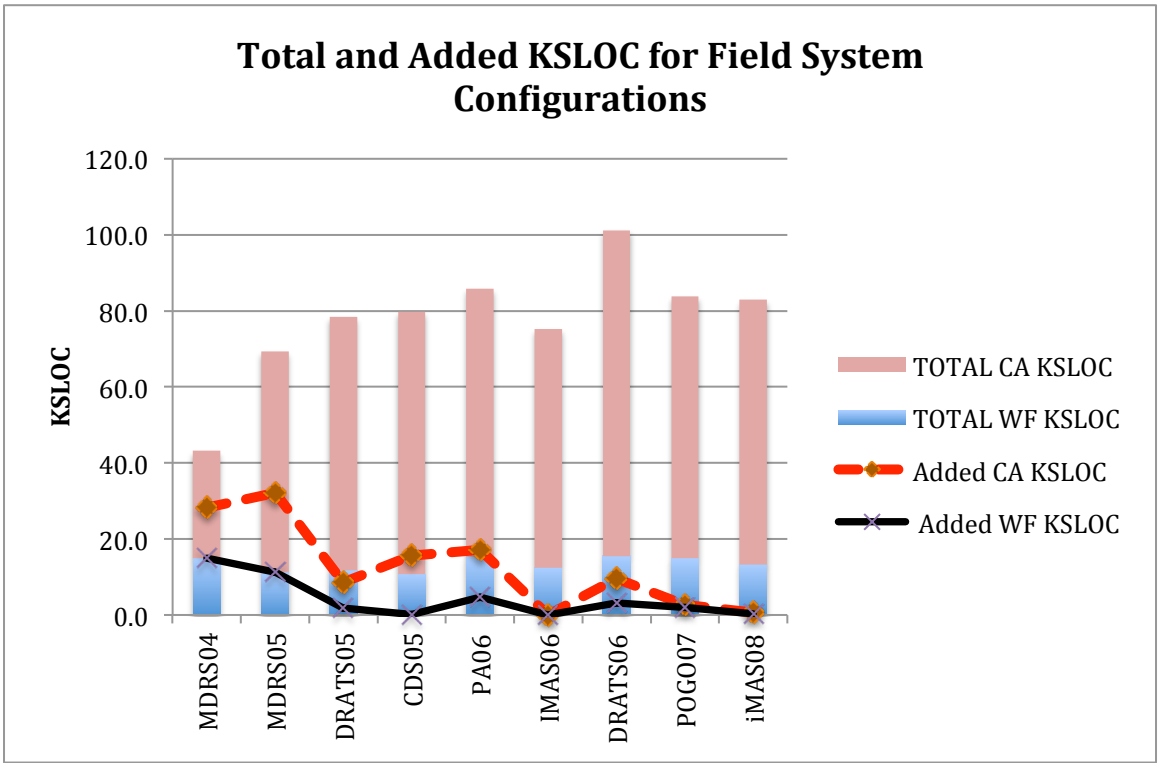

Figure 1. Total KSLOC (thousands of lines of code) for each system configuration (columns, broken into Workflow Backbone and Communication Agent parts) and new KSLOC (for new or modified agents; shown as lines). Code for Communication Agents dominates; they translate between service-oriented (task-level) messages and subsystem APIs.

Furthermore, in the aggregate *80% of added KSLOC for system reconfigurations was for task-level API translators ("communication agents"; CAs) to integrate new components* (Figure 1). But a relatively small amount was added for each configuration after the architecture was mature. On average Workflow KSLOC was increased by 14% and CA KSLOC by 13% for each reconfiguration.

Depending on the component, API translators for complex systems such as rovers might require 20 KSLOC (1 FTE or more), while simple systems such as cameras might require 2 KSLOC (.1 FTE).

The ratio of total workflow KSLOC to the total system KSLOC (Figure 2) remained surprisingly constant at 16%, which demonstrates that *the amount of code required is linear with the number of capabilities, and additions require only incremental changes to affected workflow functions.*

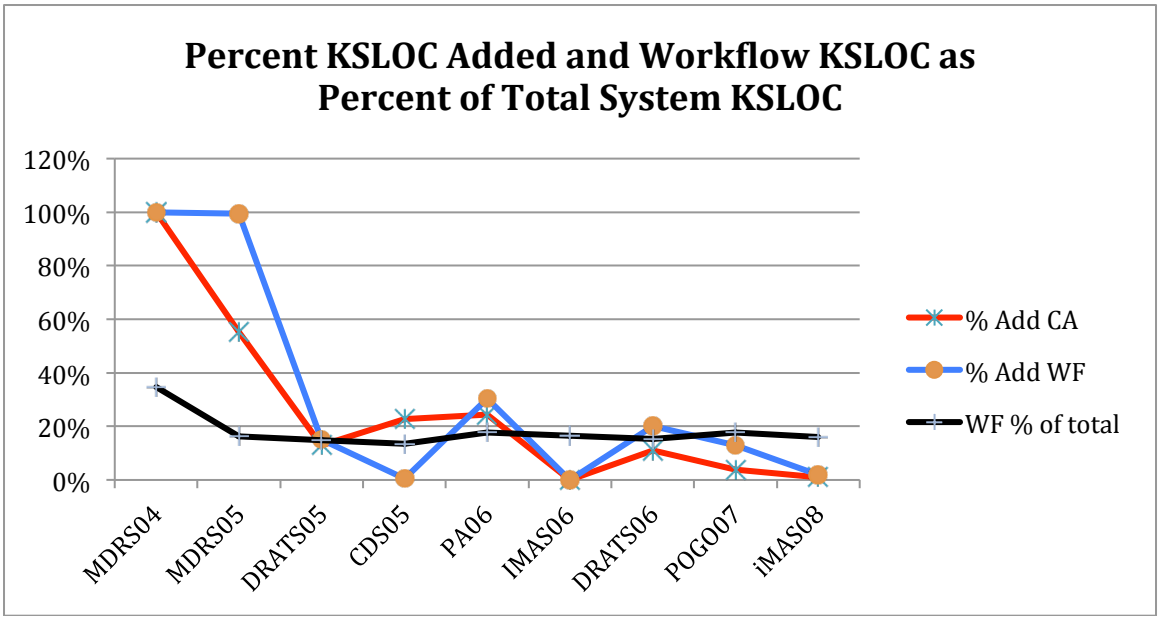

Figure 2. Percentage of KSLOC added to Communication Agents and Workflow Agents for each configuration; Percentage of KSLOC Workflow Agents relative to the system total KSLOC.

The series of exploration system experiments demonstrated efficiency in designing, developing and testing new systems. The average (and median) DDT&E elapsed time per configuration was 172 days with five programmers on average (varying from nine to one). Programming effort varied from 3.4 FTE to less than one month FTE.

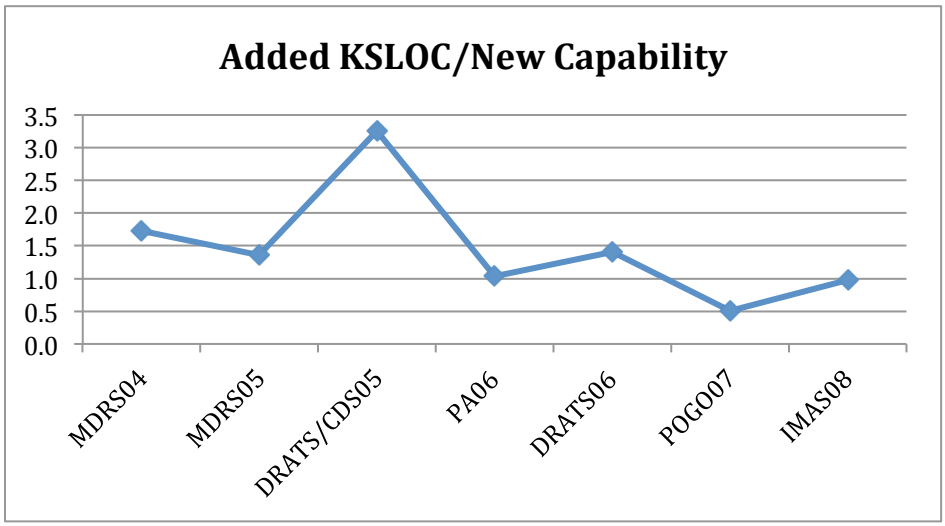

Figure 3. Additional KSLOC required per new capability (kind of information request or command).

The three key productivity findings are: *code added for each new workflow capability trends downwards from 1.5 to less than 1 KSLOC* (Figure 3); *new capabilities/FTE trends upwards from 10 to 20* (Figure 4); and *KSLOC/FTE trends upwards from 15 to 20*.

These data show that size of the modifications and effort required are generally predictable and constant, with addition of task-level APIs for new automated hardware and software systems requiring more code and time. These data suggest that the overall exploration system architecture is stable and new capabilities neither interfere with existing capabilities nor require increasing complexity of interactions. Therefore, in using an open architecture with APIs providing task-level services, we can treat capabilities abstractly and make predictions at design time of the amount of code and effort required to build or modify an exploration system configuration. Specifically, the *analysis predicts that each workflow capability added by an experienced team will require 1.5 KSLOC or less and .07 FTE or less*.

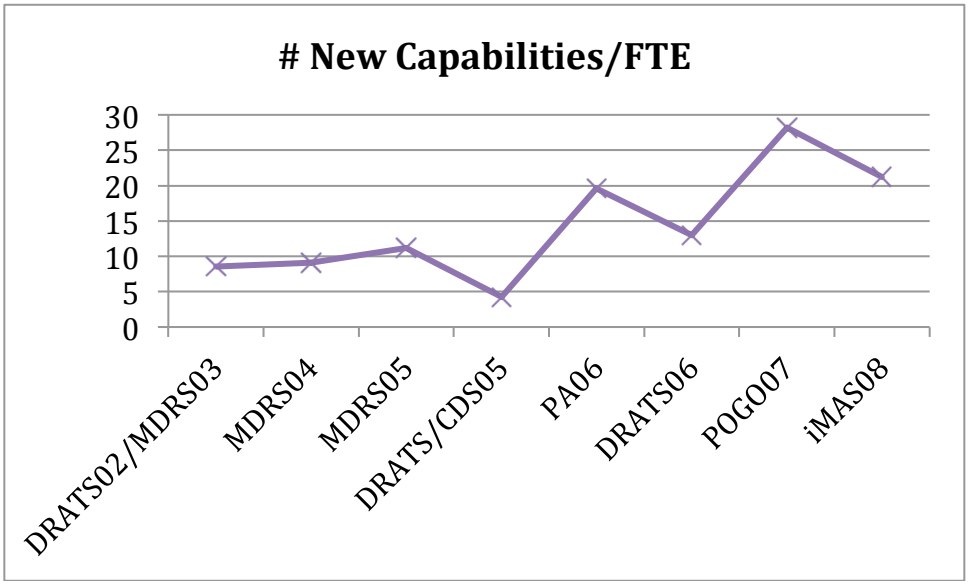

Figure 4. Number of new capabilities per Full-Time Equivalent effort (annualized over development period).

Viewing the field configurations in the aggregate, 134 capabilities were developed with 13 FTE in total DDT&E time of about 4 years. The overall average of 10 new capabilities per FTE closely fits the development efficiency of creating the mature architecture (2002-2005), when 64% of the total system capabilities were developed with 70% of the total FTE. Subsequent systems introduced relatively fewer new capabilities with increased productivity (Figure 4). Effort for DRATS/CDS05 and DRATS06 reflects significantly more complex robotic commanding for four different robotic systems.

These data show the upfront cost is relatively small given the functionality provided to the crew, with direct reuse (at no cost) of code and functionalities in very different settings. The upfront investment pays off as much smaller teams reused the existing workflow backbone, making only incremental changes to introduce completely different kinds of components (e.g., power and life support systems) with new kinds of support for crew self-reliance and safety (e.g., providing status information and alerts relevant to using resources during ongoing work activities).

1.3 Conclusions and Recommendations

An *open software architecture* enables adding, upgrading and swapping hardware and software components in an exploration system. From the perspective of LSS and human space exploration more broadly, the objective is to support upgrading and incorporating new elements (e.g., vehicles, robots, instruments, habitat modules), allowing for new forms of automation, migration and changing distribution of mission support functions, as well as new and more complex simultaneous distributed operations (Rader, 2008). This objective is often referred to in the context of "growth potential" and "incremental buildup," emphasizing technology upgrades. After analyzing a series of ten systematically developed surface systems that integrated a variety of hardware and software, we found evidence that *incremental buildup of an exploration system for long-duration capabilities is facilitated by an open architecture with appropriate-level APIs, specifically designed to facilitate integration of new components*, and this minimizes costs by reducing changes to the existing system. Specifically, the study shows the advantages of *composite APIs* that map or translate conventional component APIs (which provide access to the functional methods and data objects of components) to the language of the task (in terms of an EVA plan; names and relationships of people, places, and robots; and features of work products). This kind of API effectively enables the components (including COTS) to provide task-oriented services to the overall exploration system.

2 Introduction

This report is part of an overarching Lunar Surface Systems (LSS) Software Architecture Trade Study that identifies candidate architectures for the key software that will be used for each LSS Element (e.g., space suit, vehicle, robot, habitat). The trade study examines three main areas: minimalist architecture, EVA workflow, and real-time avionics and middleware. One goal is to estimate the level of effort and cost required for software development relative to architectural features and system capabilities.

We assume that regardless of destination and mission complexity, lifecycle cost is important, especially as it relates to facilitating international partnership, but initial costs will likely dominate decision-making. Appropriate trades might justify an upfront software investment, but the budget is necessarily bounded. Using engineering data from NASA's research in Exploration Technology and Intelligent Systems may enable generating cost data that can be used for budget planning.

This report focuses on an open architecture that provides a common goal-oriented, model-based interface that enables interoperability of a diversity of hardware and software systems whose dynamic interactions in a mobile, distributed environment create an EVA workflow automation system. The report systematically analyzes data from the Mobile Agents Project, funded by NASA's Intelligent Systems and Human-Systems Integration Exploration Technology Development Programs (2002-2006), with the objective of developing an open architecture for an end-to-end exploration system focusing on science EVAs. The objective of this report is to use the engineering data from the sequence of field experiments to recommend informative metrics for software

Open Architecture, providing the groundwork for a Phase 2 interoperability trade study using prescribed workflows in EVA scenarios.

2.1 Integrating Components by Converting them into Services

Recent work in interoperability has suggested the advantages of a *service-oriented architecture*, in which systems interact not by invoking each other directly or only exchanging data, but by requesting an abstract function (service) to be accomplished. In the agent-based open architecture described here, components (subsystems) are integrated by converting them into *services* to the exploration system (for example, see Figure 14).

More specifically, the components are effectively converted into agents so they can communicate in a common messaging scheme at the task level with other agents that manage the workflow of information requests, alerts, and commands among people and systems (referred to in this report as the "workflow backbone"). Essentially, a secondary API called a *Communication Agent* (CA) acts in consort with the component's API, serving as a "wrapper" that "agentifies" the component.[2]

A CA must implement certain generic agent lifecycle methods so that it may be properly controlled by the agent hosting environment: initialize, start, pause, resume, stop, and reset. The other methods provided by the CA are oriented around the "speech acts" in which requests and information are conveyed among agents and ultimately among the subsystems that provide the services the agents require to accomplish their goals. Speech acts specify the goals the agents must accomplish; for example, an astronaut might ask, "Where is the next activity?" which an agent will answer by consulting the EVA plan and its model of the astronaut's current location and activity. An agent might pass on the entire request or decompose it into parts that other agents are specialized to handle.

In the context of an EVA workflow system, the workflow backbone of agents will be relatively static (e.g., an astronaut's personal agent, the navigation assistant, the plan assistant), while the attached external systems (e.g., robots, instruments, databases, biosensors) will vary from one EVA to other. Dynamic reconfiguration for an EVA is possible by including the external system in the network of communicating systems, such that its presence makes new services available within a framework of existing EVA capabilities. For example, the command "Scout, take a picture of Astronaut 2" would use whatever camera is currently configured on Scout. Such systems are inherently robust by handling interactions at the work domain level. For example, if the location of Astronaut 2 cannot be determined, an agent could ask the location, which might be provided by a person or other system (e.g., "90 degrees at 20 meters from your current heading" or "at the location of the last activity"). We discovered that natural language commanding, which makes it easier for people to operate systems through voice, has the important side effect of facilitating interoperability between subsystems as well.

[2] Terms are further defined and explained in the Glossary, Appendix VI. For a more detailed explanation distinguishing agents and services and comparison of the Mobile Agents SOA to related approaches, see Appendix I.

In developing the agent-based workflow system studied here, changes were made based on field experiments that used a variety of existing hardware and software systems for both field science and routine habitat operations. The key architectural goals addressed:

1) *How mobile agents find each other on the network* (design shifted from "proxy agents" with fixed network addresses to a centralized and finally to a distributed directory service in the "Collaborative Infrastructure" [Clancey et al. 2010])
2) *How speech acts are represented* (shifted from a simple "object" with attribute to a formalized SpeechAct object that generally followed the FIPA protocol to a formal FIPA implementation in "Communication Acts"; see "FIPA" in Glossary).
3) *How the agents communicate with the external world* (e.g., remote science team) *and local astronauts not participating in the immediate work tasks* (shifted from one platform handling all transactions to including a habitat platform, "HabCom," that mediated between the mobile workflow agents and the science database, local computer display interface, and external communications).
4) *How asynchronous agents manage multiple, simultaneous requests* (shifted from strictly serial operation to managing tasks through specialized "plan assistants").

In the mature software architecture described here, called the *Mobile Agent Architecture* (MAA) these methods and representations permit distributed, mobile agents to handle simultaneous goal-oriented requests on a non-reliable network.

2.2 Analysis of Field-Tested Workflow Configurations

The study proceeds by synthesizing engineering data and lessons learned from field-tested EVA-prototype configurations using an evolving Mobile Agent Architecture over eight years (Appendix I and Appendix II). This section provides an overview to the field experiments; the next section describes the different field configurations; subsequently the data are analyzed to quantify the advantages of the architecture.

As explained below, the field experiments were research projects that explored the use of agent-based system integration for directly assisting crew members in a simulated surface mission. Put another way, the project explored how existing components (robots, cameras, computers, biosensors, GPS devices, electric power systems, etc.) could be made interoperable through a combination of agents, APIs, and voice commanding.

It is worth noting upfront that voice commanding is not just another interface modality, which is interchangeable with a conventional tabular or graphic display. Rather by virtue of the descriptive specificity of natural language and its on-the-spot availability, voice commanding provides a distinct way of getting information and controlling systems.[3] In particular, the use of voice commanding led to the discovery that requiring systems to respond to speech acts from astronauts provided a common, goal-oriented interface (using a model-based language expressed in terms of the tasks, activities, and objects of

[3] Natural language commanding can of course be provided by a visual computer interface in which one types in text or builds sentences from menus, but it lacks the always-on, "in the air" access provided by a speech interface.

the work domain) by which the EVA exploration system developers could integrate new components to interact with existing systems. Put another way, the goal-oriented, model-based language required for voice commanding serves as a layer of abstraction that unites the diversity of hardware and software (implemented in different languages and using different operating systems) that are located on distributed, mobile computer platforms.

The series of field experiments from 2002-2006 enabled exercising and measuring the architecture's capability for efficiently configuring new exploration systems. The data analysis section evaluates a variety of hypotheses in terms of how software agents were modified (e.g., direct reuse or number of lines of code added) and personnel effort (e.g., size and distribution of teams). These hypotheses relate to advantages of the open architecture for phased development of LSS capabilities:

- Ability to incrementally add new capabilities that satisfy requirements for crew self-reliance, safety, and productivity to an exploration system with existing components:
    - Ease of adding robotic systems in the same production line
    - Ease of adding a new robotic system by adapting existing software
    - Ease of incorporate existing capability for different purpose (e.g., voice mail created from voice annotations)
- Ability to reconfigure the exploration system for a different work context:
    - Ability to add new kinds external systems (e.g., science instruments, electric power systems, life support systems & biosensors)
    - Ability to directly use existing services without changing subsystems (e.g., email alerts)

Cost for these changes is estimated and correlated by counting functionalities (workflow capabilities), source lines of code, and programming time comparatively in a series of system reconfigurations.

Evaluation of an architecture must be with respect to operational requirements that require components to interact; otherwise the architecture with the lowest cost with simplest incremental buildup is trivially a system with components that can't interact. In particular, consideration of workflow functionalities provides a way of assessing the power and affordability of an architecture. A proper trade study would compare an agent-based architecture to other open architectures that provide workflow communications (i.e., command and information exchange services). In the absence of comparative data, this report focuses on describing the functionalities developed and using them to illustrate how the agent-based architecture is designed and operates. The available data, outlined above, enables quantifying the amount of code, the reuse of code, and some extent the required effort for creating workflow capabilities.

Each field test configuration implemented an integrated collection of automated capabilities, such as biomedical monitoring, robot and instrument control, habitat system monitoring, and science data collection. These capabilities are described in the section that details the field configurations (Table 1). The capabilities address typical operational

requirements pertaining to productivity, crew safety, and crew self-reliance. Security is not addressed in these field tests, however it is a key requirement of OCAMS (Figure 21; Clancey et al. 2008), a mission operations system using the MAA with the addition of the *Collaborative Infrastructure*.

3 Description of System Configurations and Data Analyzed

The data analyzed in this report consists of four different kinds of configurations or "product lines" comprising ten systems designed, developed, and tested by NASA Ames and JSC from 2002 through 2008:

- *"Automating Capcom" Configurations:*
  - **DRATS 2002, MDRS 2003, 2004** (all using EVA robotic assistant)
  - **MDRS 2005, DRATS05** (Scout vehicle), **CDS05** (K9 and Gromit robots)
  - **DRATS 2006** (pressurized suits, JSC ExPOC, and GeoPhone Array)
- *"Power Agents" Configuration:* **MDRS 2006**
- *Metabolic Advisor Configuration:* **POGO 2007**
- *iMAS Scientist's Field Assistant:* **MMAMA 2008** (HI, NM, BZE)

The "automating Capcom" configurations presumed an EVA scenario with time-delay that required astronaut self-sufficiency in carrying out EVA plans according to location and duration, storing correlated data (e.g., associating photographs with EVA activity, astronaut, location, time, and related data such as a voice annotation), and monitoring personal health, life support, and other system resources. These capabilities are described further below.

The software developed for the Automating Capcom product line includes the Collaborative Decision Systems (CDS) Project, funded by ESMD (spacecraft autonomy) and SMD (autonomy and operations). This field configuration, directly adapted from MDRS05 and using the same code base as DRATS05, was designed to demonstrate that spacecraft autonomy may require human operations, and there is therefore a place for an agent system that integrates spacecraft with databases and human work flow. The report concluded:

> [CDS 2005] focused on pragmatic ways of combining autonomy with human activities and capabilities. One perspective is that there will always be a combination of automated and human-controlled operations, through interfaces for local astronauts in the habitat or remote mission support teams. Thus, one should not focus on the particular aspects that we have automated or require operator intervention. Rather, our point is to define a simple example of such a combination and how it might be implemented using a variety of planning, voice-commanding, and visualizing systems. (Pedersen, et al., 2006)

Three quite different EVA system configurations were directly adapted from the 2005 configurations for getting information about a habitat power system (MDRS 2006), for

monitoring and understanding the relation of personal metabolic performance and life support resources (POGO 2007; Human Research Program funding), and for stand-alone data surveys on an EVA (iMAS 2008; Science Missions Directorate funding). These systems are derived from each other chronologically described further below (Table 3). Configuration diagrams appear in Appendix II.

The sources of data analyzed include:

- Code repositories (snapshot as of completion of field work)
- Highlight reports, papers, presentations, and diagrams
- Project schedules, email, budgets
- Expedition records

Source code snapshots are available starting with MDRS2004; earlier systems have a somewhat different architecture for distributed communications, which became mature in MDRS05.[4] The data about different system configurations and personnel have been organized in spreadsheets, as outlined in Appendix IV.

3.1 Categorization of Workflow Automation Capabilities

Exploration system configurations can be viewed structurally in terms of components (hardware, software) and agent software, and functionally in terms of workflow capabilities (or functionalities) provided to the astronaut crew for controlling and monitoring subsystems and EVA-related tasks, with alerting. Categories of system workflow functions are shown in Table 1 with brief explanation of what is included. The actual capabilities for each system are summarized in Appendix III.

Table 1. Categorization of EVA Workflow Automation Capabilities

Hardware Integration	Robots, cameras, sensors, instruments, displays, etc.
Software Integration	Software incorporated as separate components, e.g., planning system, Excel
Astronaut Health Monitoring	Available data and alerts, e.g., heart rate
System Health Monitoring	Computer, Power, & Life Support systems
Location Tracking	All aspects of logging, tracking, finding assets in the field
Human-Robot Coordination	Commands involving robotic systems
Plan Management	Getting status and changing the work plan
Science Data Logging	All aspects of data collection during EVA
Voice Mail	Crew communication via recorded messages
Alert Management	Control of alert types and modality
Voice Command Controls	Control of voice interface

[4] Additional data are available for the OCA Management System (OCAMS) deployed at JSC in MCC since July 2008 using the same agent-based systems integration architecture, but are not analyzed in this report. OCAMS received the JSC Exceptional Software Award in June 2010.

Workflow automation capabilities exemplified by those listed here facilitate crew self-sufficiency, safety, and productivity; in turn these capabilities are facilitated by a software architecture that facilitates communications between subsystems in the language of the task domain (Section 2.1). The capabilities of the workflow automation are demonstrated here qualitatively, through the descriptions of the system configurations.

It should be noted that the systems being analyzed were developed to illustrate typical functionalities that are useful to scientist-astronauts during EVAs in which real-time communication with mission support is not possible due to time delay. The voice commanding capabilities were designed especially for: 1) creating documented EVA products, 2) by executing the EVA plan, keeping it on route and schedule, 3) while remaining aware of logistic/safety constraints and limitations.

The MAA research objective focused on developing and demonstrating an open software architecture that enables workflow automation with COTS components (e.g., a camera, spreadsheet software) and previously developed exploration system components (e.g., the K9 rover). The systems were developed to experimentally test under authentic exploration conditions whether people would find it advantageous to make requests for information and actions in natural language (i.e., as speech acts[5]) using voice commanding. Hence the system configurations focus on developing a range of useful capabilities with a range of existing hardware and software components, rather than creating a complete, recommended configuration and capabilities.

All capabilities were introduced based on direct, empirical experience with field scientists, our experience living in the simulation habitat (for Power Agents configuration), and Apollo experience (for both science data collection capabilities and POGO07, the metabolic rate advisor). For example the alerting capabilities focus on obviously useful information (e.g., astronaut's backpack computer battery is low; the diesel generator has stopped charging the habitat's batteries) and illustrate how different kinds of information can be integrated from different sources for generating context-sensitive alerts and advice (e.g., emergency walk back route advice that relates the astronaut's current location, metabolic rate, consumables remaining, and the terrain). By contrast, other research and development efforts might focus on a particular subsystem,

[5] Speech acts are utterances in the language of the task domain, occurring in everyday speech as well as a semi-formal commanding language, with implicit assumptions about the desired response. Conventionally, "Can you pass the salt?" is not an information request, but a command, "Please pass me the salt." Commanding a robot, "Go to Waypoint 2" might mean by a convention adopted by the crew, "…and wait there until you are told to move again"). All commanding, whether by spoken language or a display menu with formal operators and operands, involves implicit meanings known to designers and system operators. The notion of speech acts becomes salient when developing a system for voice commanding workflow operations, where goals and constraints are not necessarily explicit (however agents may ask for clarification when necessary). As another example, "Associate voice note with the last image" means to store "the voice note I just recorded" in the science database (accessed via the HabCom computer), linking the voice note to "the last photograph downloaded from my camera" in the context of "me, my current EVA activity, the current time, and this location." To document this implicit character of communications between people and among agents, the agent-based architecture described here packages messages between the spoken language interface and agents using the Speech Act formalization of the FIPA standard.

for example to show how to monitor all safety rules and engineering constraints while driving an unpressurized vehicle (e.g., Scout).

3.2 Exploration System Component Configurations and Product Line Relations

Table 2 provides an overview of the EVA system configurations according to number of computer platforms running workflow agents, number of functionalities (as categorized by Table 1), robotic systems included, and other integrated subsystems. For details about subsystems, see Table 3, which also mentions how the architecture evolved. Functionalities here are grouped by kind of request per subsystem; for example, requesting the status of the life support scrubber, feedwater, or inlet temperature counts as one capability. Other examples of a single capability are: being able to turn on or off a system and to pair-wise associate voice notes, sample bags, images, and locations. Thus in general a capability was realized as multiple grammatical forms and of course an unlimited number of specific utterances naming particular objects, people, places, activities, and times.

Table 2. Mobile Agent System Configurations
Systems are listed chronologically by field test; "agent platforms" are laptop computers running agent systems with the Brahms Virtual Machine; Functions include all categories of voice commands and alerts, see **Table 1**; external systems are any devices or software with APIs communicating with agents, e.g., biosensors, cameras, email, RIALIST, Xantrex inverter, including robotic systems; see **Table 3** for further descriptions.

SYSTEM	**FTE**	**# Agent Platforms**	**# Func**	**Robotic Systems** (adapting ERA CA)					**# External Systems**
				ERA1	ERA2	SCOUT	K9	Gromit	
DRATS02	2	1	3	X					4
MDRS03	1.4	4	29	X					7
MDRS04	2.8	4	53	X					10
MDRS05	2.9	5	82	X	X				13
DRATS05	.9	4	77			X			14
CDS05	.9	4	80				X	X	11
PA06	1.1	5	45						10
iMAS06	.1	1	51						5
DRATS06	.7	5	91			X			16
POGO07	.3	1	63						7
iMAS08	.05	1	50						5

To make clearer how the systems were derived from each other, each has a "product line code." Thus the systems numbered "1" in Table 3 were developed sequentially in the order shown, 1A was adapted to produce 1B, from which 1C was produced, etc. The systems numbered "2" represent a substantially different configuration (i.e., integrating habitat power systems instead of EVA robots and devices), but were nevertheless adapted from the deemed mature MDRS05 architecture and specific agents in that configuration.

Specifically, in the second product line iMAS06 is identical to PA06, but configured for a scientist working on an EVA off the wireless network. That is to say, all of the science data logging, location tracking, and plan management capabilities (Table 1) developed for MDRS05 are contained in the second product line, but are only meaningful to use in the EVA context (with GPS and camera attached). Accordingly, in the Power Agents configuration (with inverter and power channel monitors attached, but no GPS), one could ask "Where am I?" and the system would indicate it is unable to determine your location. And in the iMAS06 configuration one could ask "When did the generator come on line?" and be told that no generator data is available. Because of the functional decomposition of agents introduced in MDRS05, many modules are carried over unchanged (e.g., from MDRS05 to PA06) and changes are almost always incremental.

To continue the description of product lines, number three, which is represented only by DRATS06 (a collaboration with Frank Delgado, Susan Tourney, and Joe Kosmo from JSC, Houston; see Clancey, et al. 2007) constitutes again a substantial change. It is derived directly from CDS05 (thus contains all of DRATS05), but now includes workflow agents running at a remote site (ExPOC in Building 30 at JSC, Houston), which enabled a controller to use voice commands to control a geophone array deployment device operated by the Scout rover. Integration with the rover and pressurized suit audio system was much improved from DRATS05, enabling all of the science collection functionalities from MDRS05 to be exercised successfully, with additional capabilities to remotely control the rover (e.g., "Scout, go to waypoint 2").[6]

Next, iMAS06 from the second product line was reconfigured to create POGO07, providing life support and consumables alerting using data from metabolic rate algorithm, which was itself integrated with a biosensor system and suit life support system. POGO was used by an astronaut in a pressurized suit supported by a partial gravity harness. In the harness none of the science collection capabilities are meaningful; instead only the astronaut and system health functions are used. Once again, inherited voice commands will be recognized, as the vocabulary grows incrementally, but without corresponding subsystems (and their agents) attached, the system responds gracefully to the lack of data.

Finally, iMAS08, a practical system for gathering field geology data at lunar analog sites, took POGO07 into the field, with science data collection functions improved (e.g., using beeps rather than verbal confirmations and automatically making certain data associations between photographs, location, and samples). The metabolic rate subsystem was omitted, because the biosensors were unavailable and would be cumbersome for practical use. (An offshoot of iMAS08, not described here, included a terrain modeling and EVA route planning tool, whose data could be integrated with the metabolic rate information and data about consumables, Johnson et al., 2009; Johnson, forthcoming).

[6] See http://www.youtube.com/watch?v=fTTrFDR9I1I, which includes video excerpts from NASA Public Affairs Office.

Table 3. Product line "inheritance" relations of MAA configurations.[7]

SYSTEM	Product Line Code	Description and Explanation
DRATS02	1A	Initial system build, single laptop platform, camera, biosensors, voice. Agent communications represented as Brahms Objects. Includes "proxy agents" for potentially queuing requests to agents on inaccessible platforms.
MDRS03	1B	Addition of an EVA Robotic Assistant (ERA), 2nd astronaut laptop, and habitat computer (for HabCom crew member), i.e., four distributed platforms. Agent communication uses KaOS with CORBA as transport layer and directory service across network for multiple platforms.
MDRS04	1C	First archived system; fully functional location and plan assistance. ERA follows astronauts in canyon, automated video tracking. Agent communication via KaOS/CORBA now uses FIPA SpeechAct envelope with Brahms "Communication Acts" as payloads; centralized directory service on mobile laptop (ATV) acceptable, but single point of failure.
MDRS05	1D	2nd ERA relay controlled by human operator; temperature probe; dynamic reconfiguration of ERA roles during EVA. Personal agents for astronauts and robots decomposed to create service-oriented "assistants"; Plan Assistant enables agents to handle multiprocessing (simultaneous open requests) through task list. Proxy Agents eliminated.
DRATS05	1E	Scout rover is configured to be the ERA; heads-up display. Insufficient testing time provided in field to complete integration.
CDS05	1E	Two weeks after DRATS05: Same software package as DRATS05 with bugs fixed. EVA system incorporates Gromit and K9 by adapting ERA agent; one astronaut, no HUD; HabCom interacts with EUROPA to control K9 via agent architecture.
PA06	2A	"Power Agents"; no robots; all inside MDRS habitat; completely new functionality in monitoring electric power system, including generator, batteries, solar panels. Voice mail implemented during two-week shake down test; configuration retains all science data collection and EVA management capabilities. Introduces sound beeps to provide confirmation for certain routine commands.
iMAS06	2A	Same software as PA06, but configured for one laptop operating in standalone (off-network) mode, for use by field scientist.
DRATS06	3	Reconfiguration of DRATS/CDS05 to enable second commanding console off-site (JSC's Exploration Planning & Operations Center, ExPOC); automated control of geophone deployment from Houston. Voice commanding by crew in pressurized suits with special microphones. Demonstrated autonomous driving of Scout ("Go to waypoint" "Follow astronaut one").
POGO07	2B	Based on iMAS06; integrates with Metabolic Algorithm (Excel VBA) connected to biosensors in pressurized suit during partial-gravity experiments. Language grammar rebuilt from scratch to be more

[7] Reports and video highlights are available for MDRS and DRATS field tests: http://homepage.mac.com/WJClancey/%7EWJClancey/WJCMarsSociety.html

		compact, enabling much faster compilation.
iMAS08	2C	Based on POGO07, with more automated science data logging; used in practical settings by geologists in Hi and NM and by divers with scuba gear (Belize).

To summarize, the product line relationships are of different types: *Incremental* components and functionalities (1A -> 1E; 2A -> 2C), *conversion* to a different setting and kind of functionality (1E -> 2A), *identical* software packages with different platforms and components included (1E, 2A), major *reconfigurations* (1E -> 3). These different relationships require fairly detailed analysis to understand productivity calculations (e.g., KSLOC/FTE) because the work required to transition the system from one configuration to another is qualitatively different. In particular, a distinction must be made between: *Reusing* an agent (no change), *adding functionality* to an agent, and *adapting* the agent for a different subsystem (e.g., rewriting the ERA CA twice, for use by K9 and Gromit). On the other hand, Table 3 reveals an architecture of great flexibility, particularly when one considers that the programming time per system configuration (developing agents and APIs) varied between less than 1 FTE and 3 FTEs, and often four or five distinct project teams at NASA Ames and JSC were collaborating. After the architecture reached maturity with MDRS05, new configurations were built in 3 to 5 months by 5 to 7 part-time programmers and two system designer/managers.

4 Analysis of Field Configuration and DDT&E Data

The data (outlined in Appendix IV) has been analyzed to reveal and explain patterns to extract architectural features that were advantageous for reconfiguration into new systems with increasing workflow assistance capabilities. The objective was to quantify the advantages of an open architecture promoting interoperability for DDT&E when building an agent-oriented workflow system that facilitated crew self-reliance, safety, and productivity. The three primary hypotheses are:

1. *The agent-oriented workflow system enables integrating components (e.g., biomedical algorithms, robots, databases) without redesigning them:* Workflow agents, in consort with CAs using SpeechActs, provide a means for new components to work together with existing components to provide task-oriented services, i.e., programming effort was at the agent level, not the applications.

2. *Cost/capability is not increasing:* DDT&E effort for constructing systems in new contexts (Power Agents and POGO) is at least comparable to adding capabilities in the mature architecture (MDRS05), involving effectively no effort to modify the pre-existing capabilities, i.e., cost is incrementally proportional to number of added capabilities.

3. *KSLOC/capability is not increasing:* Making a system more complex in terms of number of capabilities does not cause an exponential increase in size of code because capabilities tend to involve similar numbers of component

interactions and functionally specialized agents provide common services, so they are invoked and used in new contexts without modification.

4.1 Analysis of Changes to the Agent System

Figure 5 and Figure 6 show how the agents in the exploration system were modified for each configuration. The percentage of the total size in thousands of lines of source code (KSLOC) is categorized by code attributed to new, modified, and unchanged agents (for workflow agents and communication agents respectively in the two figures). The percentage of code in the configuration attributable to modified agents is further broken into that part which was added and the part that was pre-existing (parts may have been revised rather than simply carried over). The code added to communication agents is broken out to show that portion that corresponds to the RIALIST CA ("dialog agent").

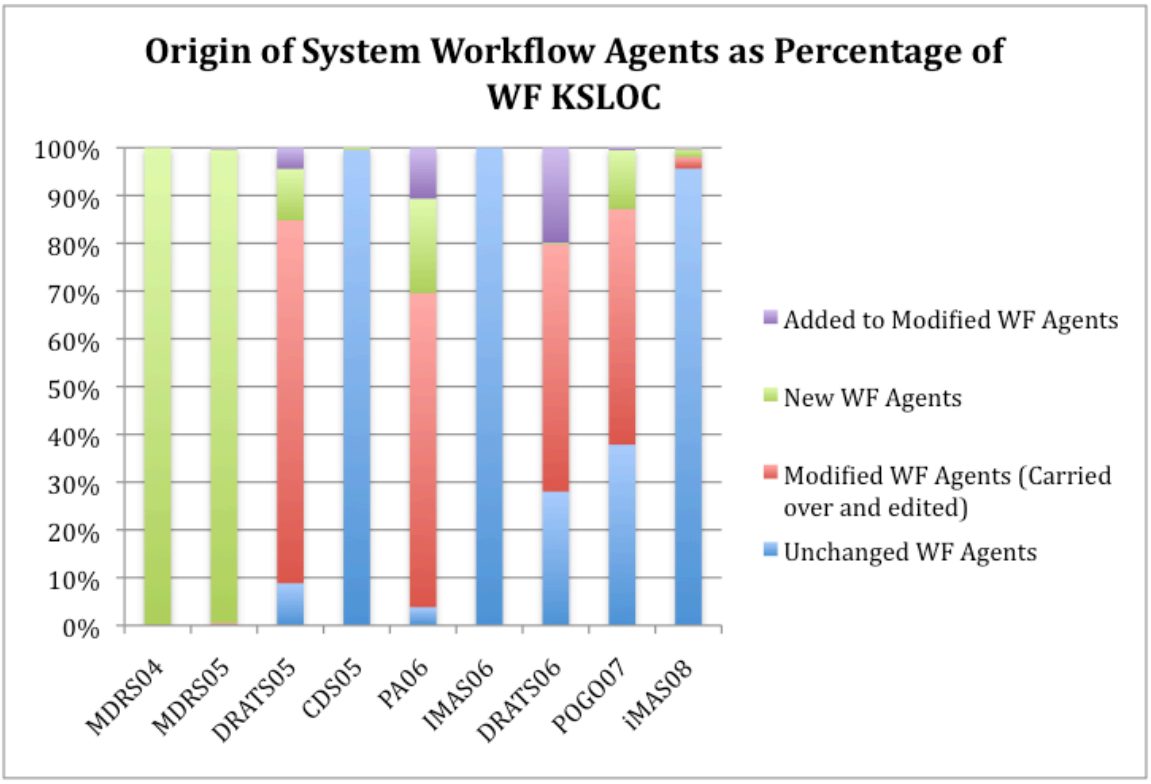

Figure 5. Reuse and additions to workflow agents for each field configuration, shown chronologically.

Figure 5 shows that a *workflow agent backbone* was established after MDRS05 (Figure 14) and was reused for subsequent configurations. Workflow agents were all restructured in developing both MDRS04 and MDRS05 (all appear as "new"), but DRATS05 workflow agent code was about 85% carried over from MDRS05. CDS05 was identical to the system used for DRATS05 because the systems were built as a single unit to operate any combination of the three robots, Gromit, K9, and Scout. DRATS06 contained no new workflow agents, and about 20% of the workflow agent network

consisted of code added to existing agents. PA06 and POGO06, systems in entirely very different settings, were similar to DRATS05 in requiring some new workflow agents for new kinds of capabilities, but otherwise retained the workflow agent backbone. iMAS is effectively a repackaging of an existing configuration for standalone use in the field, so the only coding required was for revised capabilities in iMAS08, including a new "user model" workflow agent to manage crew member preferences.

The analysis of the communication agent reuse (Figure 6) is similar to the workflow agents, with some important differences. Although workflow agents were completely restructured in developing MDRS05, *nearly half of the communication agents were unchanged.* Thus the improvements focused on the workflow functional decomposition of services to respond to speech acts, rather than the integration with the external systems. This demonstrates how workflow automation using Speech Acts for communication can be enhanced without requiring changes to existing systems or the agents that mediate with existing systems.

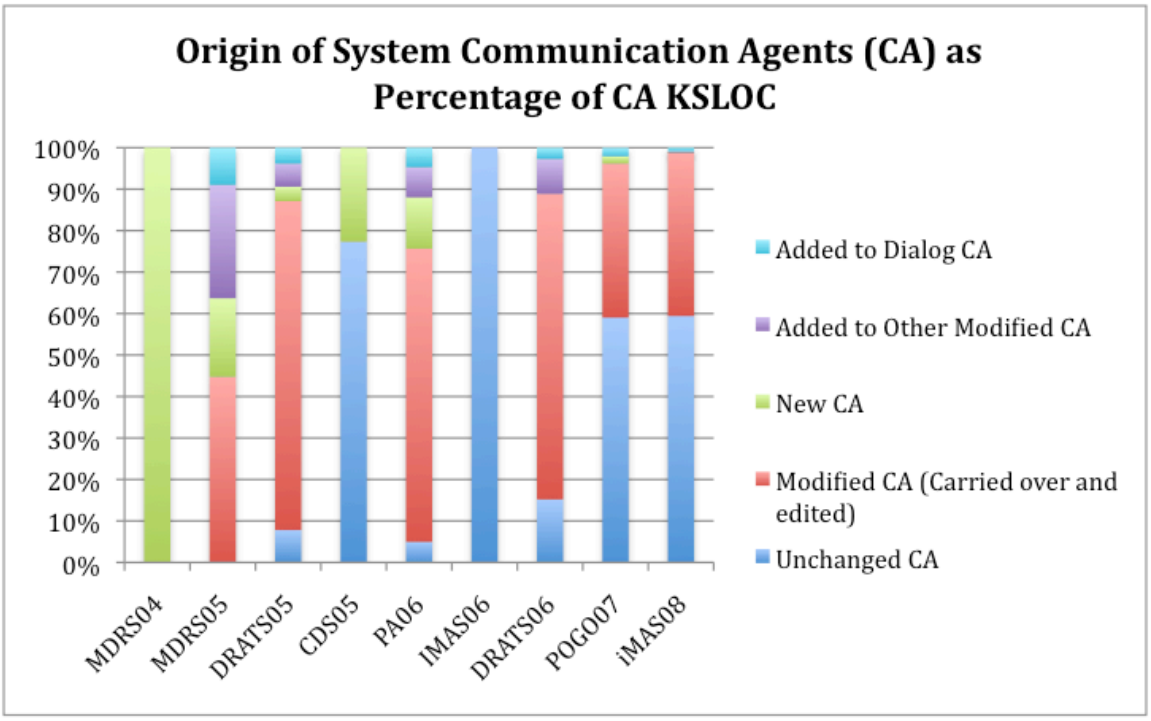

Figure 6. Reuse and additions to communication agents for each field configuration, shown chronologically. Modified agents are broken into percentage of the module that is added and portion carried over (perhaps edited). (Source code data begins with MDRS04, so all agents are categorized as "new," despite carryover from DRATS02 and MDRS03.)

On the other hand, for CDS05 no workflow agent changes were required, but more than 20% of the communication agent code was added to integrate with new external systems (Gromit and K9 robots and Europa planner software). This demonstrates how the workflow backbone can provide services to completely new components without requiring changes to the workflow agents.

Power Agents (PA06) required both new kinds of capabilities and new external systems to be integrated, so the changes to workflow and communication agents are similar. As expected, creating iMAS06 required no changes to communication agents because there are no added systems. DRATS06 on the other hand shows somewhat more modifications to communication agents (specifically Scout's version of the ERA CA), because of the new capabilities provided by Scout's Geophone deployment device, cameras, and automated drive mode.

For iMAS08 small changes to three modules (11 lines added to the Compendium database CAs 4684 lines; 9 lines to the Science Data Collector's 1056 lines; and 24 lines to the GPS CAs 571 lines) show up as a large proportion of modified code. Although measuring internal edits requires analyzing the code directly, the actual increase in module size per added capability is worth examining, as is shown subsequently.

It is useful to relate these proportional charts to the total code base (Figure 1; see Executive Summary section). This chart reveals that the majority of the code was for CAs (80% of added KSLOC), but as is evident in the preceding chart, a relatively small amount is added for each configuration (starting with MDRS05, the average addition was 14% of workflow KSLOC and 13% of CA KSLOC). Figure 2 (see Executive Summary) shows that the ratio of total workflow KSLOC to the total system KSLOC remained surprisingly constant at 16%,.

4.2 Relation of Code Size to Added Capabilities

Figure 3 relates the number of workflow capabilities added in creating a configuration to the increase in the lines of code added to workflow and communication agents combined. The data show about 1500 source lines of code were added on average for each workflow capability (as defined by Section 3.1). Overall the chart shows a decrease over time to about 1000 KSLOC/capability, perhaps because of increased stability (direct reuse) of functions provided by the workflow backbone.

What accounts for the apparent extra work in creating DRATS/CDS05? To begin, it is important to remember that DRATS05 was developed with the design configuration of CDS05 in mind, so their data are combined here. In particular, all changes to the dialog agent (RIALIST CA) appear in the source code for DRATS05, even though most were only exercised in the CDS05 configuration.

DRATS/CDS05 includes improvements based on the MDRS05 experience, including revisions to the HabCom console agent, GPS agent, and Network Assistant. But the most important changes were to the ERA CA for handling Scout (1685 additional lines), and new CAs for handling Gromit (8654 lines) and K9 (6475 lines). These additions for the three new robots account for 65% of the additional code. At the same time, not many new capabilities were introduced relative to what was added to previous reconfigurations (Table 4). In this chart and what follows, DRATS02 and MDRS03 are treated as a single combined effort, adding the 3 capabilities that were operational during DRATS02 field tests and the 26 that were completed or added during MDRS03, and adding the effort data

(FTE) for the two periods. This is justified because the initial field test during DRATS02—the first use of Mobile Agents in the field—was not sufficient to fix all of the bugs and a great deal was learned about how to configure the power, GPS, voice headset, and wireless systems to be robust under field conditions. Because KSLOC data are missing for DRATS02 and MDRS03, that entry is missing from the KSLOC analysis charts.

Table 4. Number of new capabilities for each system configuration.

SYSTEM	# New Capabilities
DRATS02/MDRS03	29
MDRS04	25
MDRS05	32
DRATS05	2
CDS05	6
PA06	21
DRATS06	9
POGO07	9
IMAS08	1

The two new capabilities introduced in DRATS05 were a heads-up display for displaying procedures on command and creation of a dynamic map displaying where data was collected (e.g., photographs) during the EVA. CDS05 included a handful of commands for moving Gromit and commanding K9 to plan a path; independently, CDS05 also introduced the capability to ask, "What is my current activity?" (to verify the personal agent's model of what the astronaut was doing).

As an example of a perhaps typical case that is easily circumscribed, POGO07 introduced 9 new capabilities with 4.5 KSLOC (.5 KSLOC/capability), which is broken down in Table 5. One new workflow agent and one new CA were added, which with corresponding modifications to the dialog agent. This ratio is clearly related to the functionality of the external system. In this case the metabolic rate advisor is a complex software program that provides a wide variety of data related to metabolic rate, life support, power, and the spacesuit, which provide four kinds of information requests and four kinds of threshold alerts, plus one integrative alert that a walkabout emergency has occurred. Obviously, how one aggregates "capabilities" strongly affects the calculation; however, the very different functions provided by DRATS06, PA06, and POGO07 suggest that the analysis is internally consistent.

The iMAS08 data provides a useful case because only one capability was added to POGO07, namely to disable/enable automatically contextually associating data in the science database during the EVA. As noted above, this change involved 44 new lines of CA code plus 596 added to the Dialog Agent and a new workflow "user model" agent of 259 lines, which yields 859, or approximately 1 KSLOC for this capability. In this case separating out handling of user preferences was desirable because two geologists were

using iMAS08 in the field (without networking), and differently configured systems were desired.

Table 5. Source Lines of Code (KSLOC) added in POGO07 configuration.

Code Modifications for POGO07	KSLOC
New WF Agent	1.8
Added to existing WF Agents	.1
New CA	1.2
Added to Dialog Agent (RIALIST CA)	1.4
Added to existing CA	0
TOTAL Added KSLOC	**4.5**

We can conclude that with experience, even when providing very different functions relevant to crew health and safety, science data collection, and self-reliance (the power system and metabolic rate advisors), the amount of code to provide new capabilities dropped and averages about 1.5 KSLOC/capability (Figure 3, see Executive Summary). Note that this does not include code changes that might have been required to component APIs or changes to internal component applications, such as a robot's operating system, because the data are not available. We do know that most if not all changes occurred in APIs because the commands provided by Gromit (e.g., move a certain distance at a certain bearing) and K9 pre-existed integration into the workflow system. COTS components (GPS, biosensors, Excel, One Channel system) were not modified at all, including their APIs, for the integration process.

The extra effort worked for the DRATS/CDS05 configuration suggests that adding a component such as a robot that provides a variety of different capabilities will require a substantially more complex CA (e.g., compare integrating a camera that can only download images to adding a robot that controls two kinds of cameras, drives autonomously, follows people, etc.).

4.3 Relation of Voice Commanding Interface to Capabilities

We should expect the amount of code added to the Dialog Agent (RIALIST CA) to be proportional to the number of capabilities added. The data indicate each capability on average required about 300 lines of code to be added to this communication agent, which passed voice commands (as Speech Acts) to workflow agents for processing. The iMAS08 single added one capability required 600 lines; other configurations such as MDRS and DRATS with 20 to 30 capabilities averaged fewer lines. The iMAS08 configuration, whose source code is being compared here to POGO07, also included improvements that do not appear as new capabilities. For example, the following responses were included as being equivalent to "yes": yeah, yup, sure, affirmative, okay, concur. If we count this as another capability, then the ratio is .3, which is within previous bounds; so the variation for iMAS08 can be ignored.

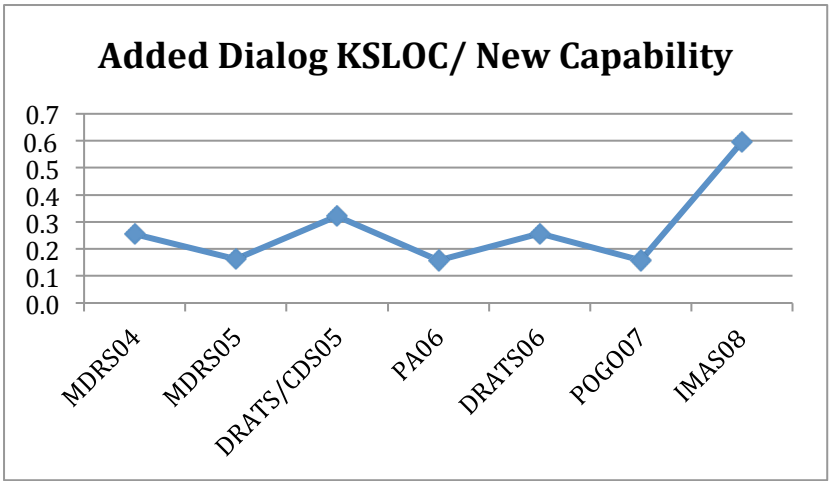

Figure 7. Source lines of code added to Dialog Agent (RIALIST Communication Agent) for each Capability.

Relation of Communication Agent Code to Kind of Component

Figure 8 shows the lines of code for the communication agents for each of the indicate hardware and software components. When CAs were modified for subsequent field experiments, the chart indicates the initial and final sizes. The components are sorted with hardware followed by software from left to right. The chart indicates that CA size for hardware components varies greatly, as expected, with devices that are providing only measurements (data) and/or status information (battery monitor, camera, GPS, biosensors) have relatively small CAs, while robotic systems are relatively very large. The average software system integrated required on average smaller CAs. Compendium organizes and stores EVA data as graphic networks; Science Organizer organizes the data into a semantic database, dynamically arranged into an Explorer-style browser on web pages available on the Internet.

4.4 Productivity Analysis

This section examines the productivity (effort required) for adding workflow source code, including both workflow and communication agents (but as before, excluding APIs because those data were not archived). Although KSLOC data are complete and reliable, the FTE data are estimates and subject to large errors, as indicated in Figure 9.

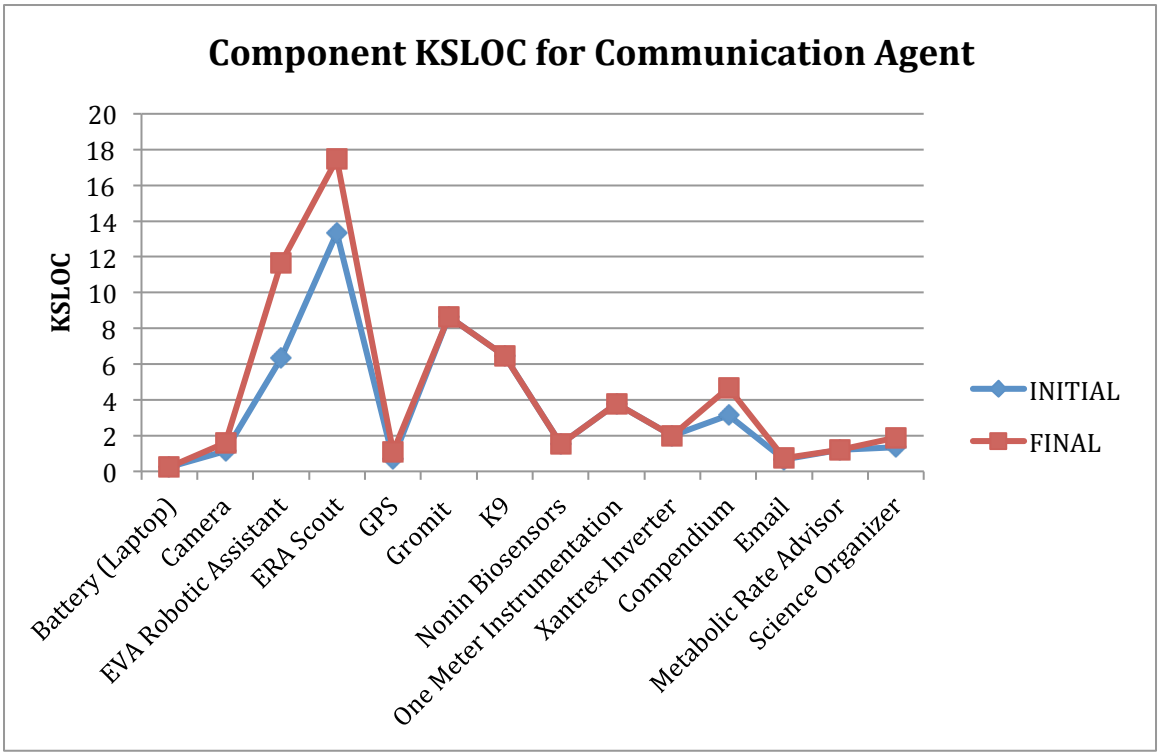

Figure 8. Source Lines of Code for Communication Agents for Integrating Hardware and Software Components.

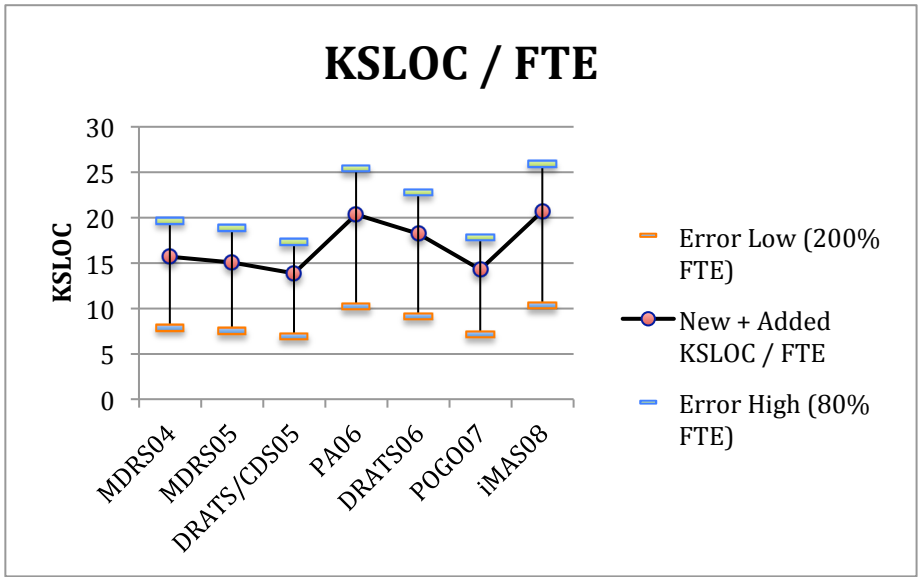

Figure 9. Ratio of Source Lines of Code for new or added to Workflow and Communication Agents to Programmer Effort (Full-Time Equivalent). Effort is annualized over the development period (i.e., effort during period * (number of period days / 365 days)). Error bars indicate range for underestimating effort ("error low," deemed to be more likely) and overestimating ("error high," unlikely).

Figure 9 shows KSLOC/FTE. The error in FTE data as depicted in the graph cautions that we not over-interpret the pattern, but it is nevertheless striking that the productivity for adding capabilities was relatively constant (though perhaps improving) in the first three configurations, as the architecture was maturing. As noted in discussing Figure 3, the establishment of the workflow agent backbone required for an EVA exploration system allowed development of new capabilities after MDRS05 to focus on the new CAs (if any) and changes to the Dialog Agent to accommodate new voice commands.

MDRS04 KSLOC/FTE productivity is calculated as if it were the first year as if all modules were created from scratch because it is the first year for which archived source code exists. In some respects, this is not a problem because we know that the source code from MDRS03 was significantly revised and reorganized. Also, in counting new capabilities and KSLOC, we appropriately exclude those that were operational as a result of MDRS03 development and testing. What is missing therefore are the KSLOC productivity data from the two earlier iterations: DRATS02 in which the personal agent of the astronaut (with science data collection, biosensor monitoring, and GPS tracking) was developed, and MDRS03 in which the personal agent of the ERA (with capability to take photographs, follow astronauts, and move on command) was developed.

The data indicate that KSLOC/FTE productivity was relatively constant for the EVA exploration systems starting with MDRS04, increasing for the last configuration (DRAT06). The apparently decreased productivity are not significant given the error possible in the FTE data. However, a slight decrease is not surprising for DRATS05/CDS05 because of the involvement of two new teams at NASA Ames who developed Gromit and K9. We previously noted that added KSLOC per new capability significantly increased for DRATS05/CDS05 (Figure 3) because of the complexity of the three robotic systems that were integrated.

An obvious correlation is that the increased productivity for DRATS06 occurs because the team added capabilities to the existing set of components developed (but not sufficiently tested) for DRATS05. Including ExPOC at JSC in DRATS06 essentially required only installing a copy of the HabCom system, which was already operating remotely via the Internet in MDRS05 (and had been configured for the Meteor Crater).

Productivity increased in developing the PA06 system and iMAS08 (which included only one new capability and no new components), but productivity for POGO07 is more similar to the EVA exploration systems. Development of PA06 might have been relatively productive because it is the only configuration in which all of the developers belonged to one organization (the NASA Ames Brahms group). In contrast, developing POGO07 involved a new collaboration with a NASA JSC team in the Human Research Program, with considerable work to understand the Metabolic Rate Advisor and develop its CA. As noted before, iMAS08 is counted as having only one new capability, so the error bar in the KSLOC/FTE ratio is high.

An important trend to consider is that *KSLOC added, number of new capabilities, and development effort are dropping over time for new configurations*, relative to the original

MDRS04 and MDRS05 systems (Figure 10). New systems and revisions after 2005 show strong correlation between number of new capabilities and added lines of code as effort required drops. This fits the claim that a workflow backbone was in place by MDRS05, such that it and most CAs were reused (with only incremental modifications) for subsequent configurations. What is striking is that *these subsequent configurations provided capabilities for commanding very different kinds of components*. As previously mentioned, DRATS/CDS05 required relatively more code for the number of new capabilities, explained by the complexity of integrating with three new robotic systems. However, the relation of KSLOC to FTE remained about the same, showing that the programming team's efficiency was relatively constant. One can also see again here (as noted for Figure 3) that the amount of new code for MDRS04 and MDRS05 per capability was relatively high compared to later systems, as would be expected because that develop involved creating (and reconfiguring) the workflow backbone.

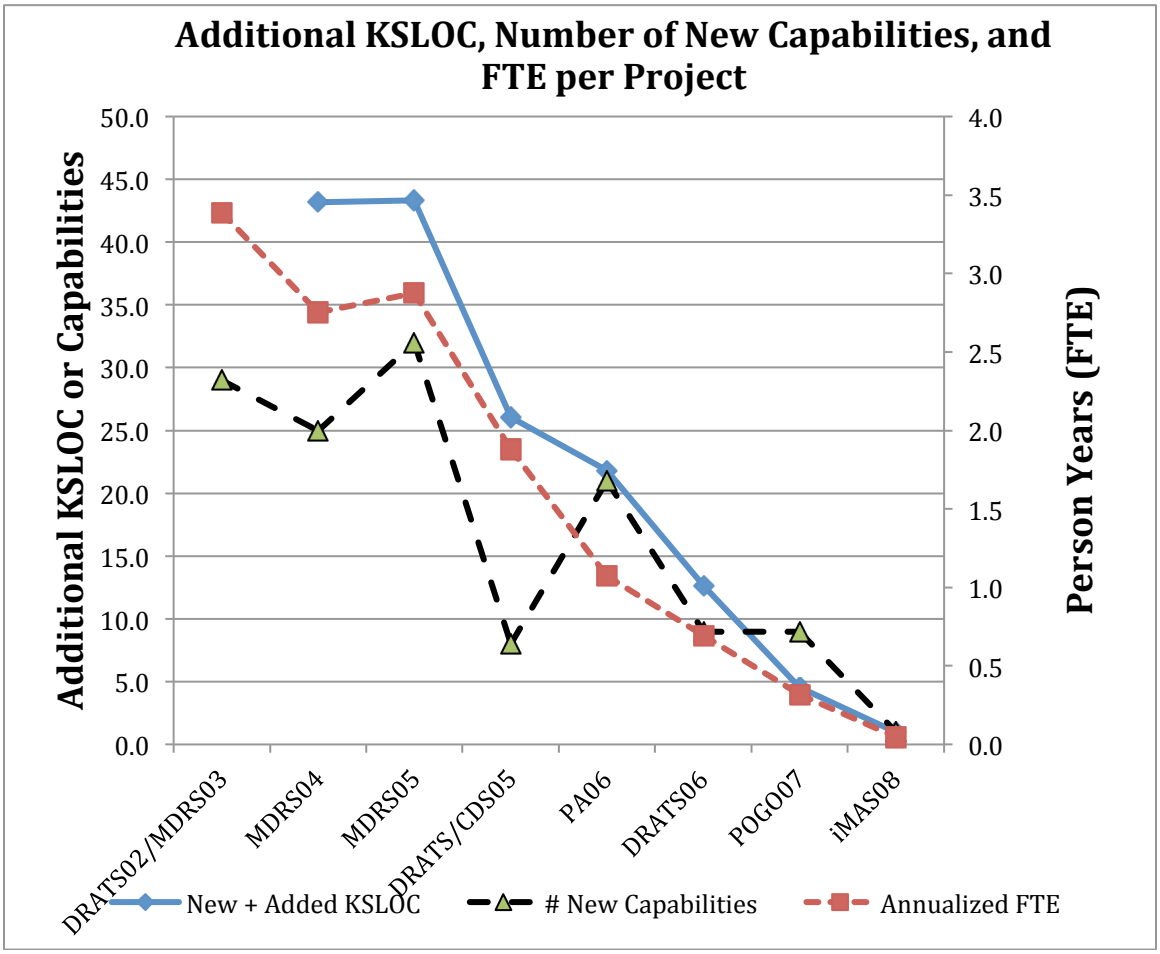

Figure 10. Additional Lines of Workflow and CA Code Compared to Number of New Capabilities and FTE.

The relation of added KSLOC and number of new capabilities (Figure 3) suggests that new capabilities/FTE should also be correlated (see Figure 4 in Executive Summary). Indeed, we see a marked improvement in productivity over time, with dips for

DRATS/CDS05 and DRATS06 possibly because of the number of people involved integrating the new systems. Figure 4 includes the number of new capabilities and FTE data from the combined DRATS02 and MDRS03 field seasons. It is not surprising that both values are at or near the maximums, fitting the claim that these are proportional (Figure 4) and upfront effort was relatively high.

The productivity of DRATS06, relative to the MDRS and DRATS/CDS configurations, is noteworthy because it involved commanding SCOUT to deploy a new instrument and to drive autonomously, and all commands could be initiated in the field by the crew, in the habitat, or remotely at JSC (ExPOC). One could anticipate very different results if adding new capabilities increased component and workflow interactions. Instead, the workflow backbone and CA architecture is designed to make capabilities completely modular, so adding new capabilities requires adding, but rarely revising code.

In summary, Figure 4 shows that after 2005 *very different system configurations*—involving a habitat power system, remote operation of Scout and its instruments from ExPOC, life support systems and metabolic rate interpretations—*were provided with increasing efficiency*. Furthermore, the same systems developed for MDRS05 and then PA06 were repackaged for iMAS08 to allow operation off the network in a standalone mode (with downloading of data after return to the "habitat")—with effectively no change to the workflow automation. iMAS08 was field tested in lunar analog sites in Hawaii and New Mexico, as well as used for surveying a coral reef (using a SCUBA system with a voice loop[8]).

Some effort was made to quantify the "complexity" of the systems in terms of number of interacting components and relate to productivity, such measures are always far courser than the count of capabilities and KSLOC. However, one basic measure of interest is the number of components integrated in the system (Figure 11).

The data show as expected that the number of components is increasing through the development DRATS/CDS05, with a slight decrease for DRATS06 because of the absence of the robots Gromit and K9 and Europa planning system. The configurations developed for other purposes —PA06, POGO07, and iMAS08—naturally have a different number of components and can be seen to be simpler. This pattern partly showed in Figure 10, where we saw the number of capabilities added and FTE required significantly decreased as well. One could argue that the reduced number of components in these systems reduces the interactions that occur in the workflow (and among the programmers working across organizations), perhaps accounting for the increased productivity in adding capabilities (Figure 4).

[8] Distortions in the articulation of SCUBA divers required that their voice commands were repeated by an operator (HabCom) on a nearby boat; however, the computer-generated replies were directly heard and responded to by the divers.

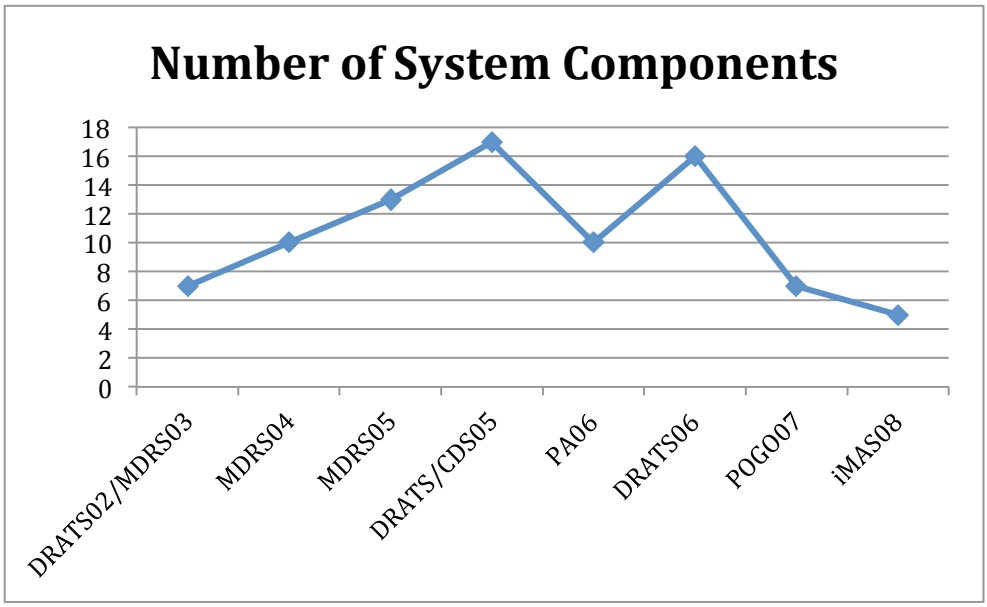

Figure 11. Number of Components (Hardware and Software) Integrated for Workflow Interoperability in the Exploration Systems

5 Lessons Learned and Recommendations

This section summarizes the lessons learned and makes recommendations for an agent-based open architecture that promotes interoperability. The fundamental conclusion from the series of ten configurations developed using this architecture from 2002-2008 is that *a workflow service-oriented approach enables increasing the number of interacting systems across diverse SW and HW components while maintaining or even improving developers' productivity.* The systems integrated included software COTS (e.g., Microsoft Office), peripheral instruments and sensors (e.g., digital camera, pancam, biosensors), automated "autonomous" machines (e.g., robotic assistant, robotic vehicle, science rover), and automated monitoring and control software (e.g., pressurized suit life support system; robot planning and control system; metabolic rate monitor). In addition, crew interactions with these systems were all accomplished through voice commanding using a spoken dialogue system (RIALIST, a research variant of the commercial NUANCE program). The workflow interactions demonstrated involved automated data integration and commanding among components; feedback managed by the workflow processes dynamically related state data and commanded the systems for goal-oriented actions requested by the crew (e.g., having a vehicle or camera follow an astronaut on EVA).

5.1 Review of Findings Experimenting with an Agent-Based Workflow Architecture

We showed that *basic support for crew self-sufficiency, safety, and productivity can be provided by workflow automation, which is facilitated by interoperability that occurs in the language of the task domain*, rather than interactions between components in the

programming language of the applications (data objects and methods) or simply via data exchange standards and protocols. An important subsidiary claim that lies outside the scope of this report is that enabling the crew to communicate in spoken language increases crew self-reliance, safety, and productivity.

The language of the task domain is in terms of *activities*, *named entities* (including places, people, devices, robotics, routes, etc.), *dynamic goal-oriented relations* (e.g., "associate" "follow" "tell me when…"), schedules, etc. The language is formalized internally as Speech Acts according to the FIPA standard for agent communications. The translation between these two levels is provided by an agent-based architecture in which specialized "Communication Agents" mediate between component APIs and a relatively stable backbone of workflow agents (e.g., see Figure 14).

Adding or replacing a component involves developing a CA for that component and possibly making adjustments to the component's API to expose operations desired for workflow automation. When the workflow capabilities are already implemented, incorporating the new component is handled by a Directory Service that allows agents to find and refer to that component by name and physical location (geographic and network address). Otherwise, adding new workflow capabilities involving human interaction requires modifying the interface and the interface's CA to appropriately receive new kinds of requests and pass them to the appropriate workflow agent. In the preferred architecture the interface recognizes and generates spoken language, but display menus, button controls, and menu/touch screen interfaces may be also used to formulate requests in the language of the task domain, and other interfaces (e.g., tones and lights) can be used for communicating information. These interfaces could be integrated with additional communication agents.

This report also showed that *incremental buildup of exploration systems, assuming workflow automation for reasons given above, is enhanced by an open architecture*. In terms of the agent-based approach, this means that workflow agents can be practically modified to communicate with new or revised CAs and provide additional workflow services in an incremental manner. Put another way, if agent-based workflow automation were adopted to provide interoperability as described above, but modifying this automation were too complex, than the architecture would not be practically open to modification and make incremental building too costly.

In actuality, the data showed that the agent-based approach, effectively converting components into agents by enabling them to provide services (through CAs) at the level of the task domain, provides a practical open architecture. Perhaps the best example is the reconfiguration for DRATS/CDS05 in which three robotic systems and a planning tool were added to the exploration system (totaling 17 hardware and software components), involving a 25% increase in the number of workflow capabilities (from 32 to 40). The source lines of code (KSLOC) added per capability were significantly higher than in the mature architecture that was being revised (MDRS05), explained by the complexity of the CAs required for the robotic and planning systems (relative to past experience in adding primarily passive instruments). The number of capabilities added

per FTE also dropped slightly. However, the KSLOC/FTE was similar to previous experience, showing that the team encountered no design or coding difficulties—the work was restricted to managing the translation between the components and the workflow system, not in modifying the workflow backbone to accommodate the new components. This example demonstrates the advantage of an agent-based workflow backbone architecture, with component systems not interacting with each other directly (as in many object-oriented systems promoted by CORBA), but through agents that providing services required by the crew. Again, notably these services are internally provided and communicated *in the language of the crew's work domain*, *not the objects and methods of the components exposed by their APIs.*

5.2 Advantages of the Information Exchange Services Layer

In summary, an agent-based workflow architecture including agents for translating between component APIs and the language of the task domain provides an appropriate level for plug-in/upgrade of components for both data and control integration. To make this kind of interoperability explicit, *it is recommended to add Information Exchange (Workflow) Services to the C3I Architecture.* Rather than having components interact only (or primarily) through Data Exchange Services, data exchange is used by the Communication Agents that translate between the language of the task domain the component APIs. This means that components need only expose data and methods required for workflow once, and the many-to-many mappings required by different workflow capabilities are handled primarily by the workflow agents in gathering required data and passing on parameterized Speech Acts to the CAs of the components involved (illustrated by the diagrams in Appendix II). These Information Exchange Services are built upon existing inter-process communication methods including CORBA and HTTP.

In contrast with methods for integration that require individual component developers to learn and speak in a common language, an exploration systems architecture with information exchange services enables the embedded components to be implemented in different languages with different operating systems. The information exchange services themselves use a common level of abstraction (workflow agents communicating via Speech Acts) that the components do not use and their API developers did not need to know. In this manner, both connectivity and information exchange operations are made independent of the components, enabling robotic and life support systems, instruments, planning and monitoring software, and the like to be developed without regard for how interoperability will be managed. The only requirement is that the components provide APIs to expose data objects and methods required to provide access to data and external control.

The CA-API translation approach might be impractical if a great deal of work were required to develop the CA for new components or workflow capabilities. In practice, the data show that the amount of code required for a component CA is correlated with the number of capabilities requiring that component (Figure 8). Robotic systems and monitoring/controlling software can be controlled/configured by crew members and also generate data (through the subsystems they control), thus providing a relatively large number of possible workflow operations that can be operated (e.g., automatically

generating and storing data on a map provided to a remote science team as robots move). Such systems generally have commands for configuring them (e.g., "take a picture of the rock called Broccoli") and alerting the crew when certain conditions occur (e.g., "tell me when the habitat power usage exceeds 15 amps"). In contrast, instruments, such as cameras, are capable of fewer operations, don't have subsystems, and don't include monitoring functions, so they have relatively few workflow capabilities associated with them and their CA is correspondingly smaller.

Other findings related to the efficiency of using this open architecture include (see Section 4):

- Adding the SCOUT rover (DRATS05) was significantly less costly than developing the original interface for the ERA (MDRS03).
- Adding the K9 and Gromit rovers with the Europa planning system required new CAs of substantial complexity (CDS05). Productivity measured in KSLOC/FTE was similar to previous efforts, but new capabilities/FTE dropped significantly, which reflects the inefficiency inherent in a new collaboration with three different subgroups.
- Correspondingly, productivity by both KSLOC and capability measures markedly improved when the Brahms team developed a system that did not involve programmers from other groups or organizations (PA06).

5.3 Advantages for Design, Development, Testing, and Evaluation

The series of exploration system experiments demonstrated efficiency in designing, developing and testing new systems. The average (and median) DDT&E elapsed time was 172 days with five programmers on average (but varying from nine to one). Average total programming effort (excluding two managers' work on scenario design and project coordination) was 1.4 FTE, varying from 3.4 FTE (DRATS02/MDRS03) to less than one month FTE (iMAS08).

The findings from the system reconfigurations suggest that *the agent-based workflow architecture is advantageous because it limits and controls inter-team interactions (coordination) required.* The design process is strictly oriented to the workflow capabilities being introduced—changes to the work activity scenario are related to new workflow functions desired. In particular, the subgroups of the development team correspond to the logical design levels:

I. Scenario designers and project managers
II. RIALIST (voice commanding) programmer
III/IV. Agent modelers/programmers (Workflow Agents = III; CA = IV)
V. API programmers for components/subsystems
VI. External system developers (e.g., robots).

Levels I-IV were co-located at Ames; Levels V/VI were at JSC or Ames depending on the robotic or other external system. Coordination was most necessary and intensive for relating adjacent levels (I-II, II-III, etc.).

For example, if an automated tool is introduced, the crew is provided with voice commanding for turning on/off the tool, configuring its operation, directing how data be configured in the ongoing EVA/science database, etc. After voice commands are agreed upon, RIALIST's grammar is modified accordingly, and Speech Acts are represented to carry the requests from the RIALIST CA through the workflow backbone and/or directly to component CAs, as necessary to process the request. If a new type of Speech Act is involved, the programmer responsible for the component CA must write code for reformulating the Speech Act in terms of the component's API data objects and methods. If the request involves a new kind of automated control or data being provided to the crew, then the programmer responsible for the component API will likely need to modify the API and possibly the device's internal operating system (e.g., this occurred with the Geophone array deployment device was added to Scout for DRATS06). In summary, modifications required are predominantly modular and narrowly defined by the representation of workflow automation as Speech Acts.

Regarding other DDT&E considerations, *the architecture has advantages in enabling scenario-based design* (specifically, a simulation-to-implementation conversion methodology, Clancey et al. 2008), human-centered design, distributed development teams, and iterative development of partitioned functionalities.

For testing, the architecture provides for single-platform integration—each developer runs the entire system of agents and components (usually as simulations or providing a simulated data stream) on one computer. After these independent, component-level tests, the team gathers for about a week of operational-readiness tests. Two to three weeks later, the team goes to the field, usually engaging in about a week of integration tests, shifting from wireless connectivity inside a habitat to experimental use in simulated EVAs, with progressively more complex scenarios (e.g., at a greater distance for longer periods).

Rapid reconfiguration enables redesign during field testing; the agent decomposition of services facilitates logging data about speech recognition, network responsiveness, message passing throughput and latency, etc. The modularity and generality of the agent workflow backbone and CAs was especially well demonstrated by the introduction of a voicemail capability during the MDRS06 field work. Essentially, the voice annotation capability for describing science data, which included capabilities to record and replay voice notes, was adapted to play a voice note not only on direct request but for a different crew member at a prescribed time. This required introducing two new agents, the voice mail client and server (Figure 18). The voice mail capability was designed by the team, implemented by one person, and in operation within a few days—and these changes occurred to the Power Agents configuration in the middle of the two-week field test.

5.4 Advantages for Reconfiguration Efficiency and System Size

The three key ratios— amount of code added for each new workflow capability, new capabilities/FTE, and KSLOC/FTE—show that size of the modifications and effort required are generally predictable and constant (with addition of CAs for new automated

hardware and software systems requiring more code and time). These data suggest that the workflow architecture is stable and new capabilities neither interfere with existing capabilities nor require increasing complexity of interactions. Therefore, *in using an agent-based workflow architecture with Speech Act communications, we can treat capabilities abstractly, and make predictions at design time of the amount of code and effort required to build or modify an exploration system configuration.*

Viewing the field configurations in the aggregate, 134 capabilities were developed with 13 FTE in total DDT&E time of about 4 years. The overall average of 10 new capabilities per FTE closely fits the development efficiency of the workflow backbone (through MDRS05, see Figure 4), when 64% of the capabilities were developed with 70% of the total FTE. Subsequent systems introduced relatively fewer new capabilities with increased productivity. Effort for DRATS/CDS05 and DRATS06 reflects significantly more complex robotic commanding for four different robotic systems.

These data show perhaps an upfront cost that is itself rather small given the functionality provided to the crew, with direct reuse (at no cost) of code and functionalities in very different settings. The upfront investment pays off as much smaller teams reused the existing backbone, making only incremental changes to introduce completely different kinds of components (e.g., power and life support systems) with new kinds of support for crew self-reliance and safety (typically here, providing status information and alerts relevant to using resources during ongoing work activities).

On average, after the stable backbone was developed for MDRS05, each configuration added about 13% to the code base (Figure 1), with 80% of the added code attributed to component CAs. The ratio of total workflow KSLOC to the total system KSLOC remained surprisingly constant at 16% (Figure 2), which is perhaps another way of demonstrating that *the amount of code required is linear with the number of capabilities and additions require only incremental changes to affected agents*.

5.5 Lessons Learned from Field Experiments

The development of the workflow agents involved a substantial learning process through trial and error in the field experiments, particularly during DRATS02, MDRS03, and MDRS04. We found it necessary to explain to other DRATS participants that the Mobile Agents system should be viewed as being more akin to a word processor than a document; they were focused on particular capabilities (and gaffes) than the generality and flexibility of the agent architecture to integrate and access/control a wide variety of hardware and software. One observer of the inaugural trials of the prototype system remarked, “Not ready for prime time,” suggesting that he viewed DRATS as an opportunity to test systems, to demonstrate operational readiness, rather than to experiment and learn.

The methodology of empirical requirements analysis employed during the Mobile Agents project (Clancey et al., 2005) involved using the prototype system for authentic scientific exploration (e.g., use by scientists in a new terrain that addressed their specialized expertise and interests). During these experiments we frequently discovered the value of

additional workflow automation (e.g., during iMAS08 it became obvious that explorers wanted to ask sometimes "Guide me to <location>"—receiving alerts thereafter to correct course—rather than repeatedly asking for the distance and bearing to the desired location). We recognized requirements and invented methods to make the wireless, distributed agent communications more robust and to handle loss of communications gracefully. And we made many improvements to the interaction between people and their personal agents and robotic assistants to make commanding more reliable and simpler (e.g., substituting a beep confirmation for non-critical requests; not having an agent speak when you are speaking to someone else). We also discovered some complications that will require future research and experimentation (e.g., how to avoid having an agent speak to you when you are listening to someone else).

Some of the lessons learned about what services workflow agents should provide and how they should be structured include:

- **Use of a distributed directory service** (CI; Clancey al. 2010) to deal with unreliable wireless communication and to allow subsystems to be disabled and restored in a running system configuration (a runtime form of "open architecture").
- **Use of a workflow backbone** consisting of individual personal agents for people and robots, complemented by functional workflow agents for navigation, planning, database management, communications (e.g., via email, GUI, voice loop, voicemail), distinct from the Component CAs that interface with component APIs provides great flexibility for reconfiguring components during an expedition for different EVA requirements (e.g., the transformation of MDRS05 to DRATS05 and CDS05; MDRS05 to PA06; PA06 to iMAS06, POGO07, and iMAS08).
- **Use of a web-based semantic database** to consolidate data from different sources for a common interplanetary repository (Berrios et al. 2007), using persistent queuing and repeated transmission until acknowledgment of delivery (fault tolerant if platform reboots).
- **Experimentation with a variety of reconfigurable, mixed communication methods** for controlling and getting data from arbitrary systems: Voice (including shared loudspeaker), menu-based GUI, audible tones, heads-up display of charts, internet web pages, and email.
- **Allowing alternative data-exchange services** (e.g., SOAP, OAA) for communications between CAs and component APIs.
- **Providing methods for creating and relating different types of data for different purposes** (e.g., photograph files, GPS coordinates, time stamps, biosensor and life support system telemetry, alert thresholds, voice recordings, terrain maps).

5.6 Conclusion

We showed that an agent-based architecture with task-level message passing (Speech Acts) has measurable advantages for enabling workflow interoperability and efficiently

making incremental modifications. Ideally, this software architecture would be compared in a trade study to alternatives that promote integration of arbitrary hardware and software components. However, we do not know of any competing architecture for which comparable field configurations have been developed, let alone with data permitting comparison. The adoption this agent-based architecture by JSC Mission Operations Directorate for workflow automation in Mission Control (OCAMS, Figure 21; Clancey et al. 2008) and its receipt of the JSC Exceptional Software Award in 2010 shows that the architecture is mature and useful for flight operations. Indeed, the DRATS06 configuration with ExPOC at JSC showed that use of this architecture in LSS field tests would enable immediate integration between remote and surface work activities.

6 References

Berrios, D. C. Sierhuis, M.. and Keller, R. M. 2007. Geospatial information integration for science activity planning at the Mars Desert Research Station. In A. Scharl and K. Tochtermann (eds.) *The Geospatial Web: How Geobrowsers, Social Software and the Web 2.0 are Shaping the Network Society*. Springer-Verlag, London, pp. 131-140.

Bridges, C. P. and Vladimirova, T. 2011. Real-time agent middleware experiments on java-based processors towards distributed satellite systems. *IEEE Aerospace Conference 2011 (IEEEAC'11),* Big Sky, USA, paper #1639.

Clancey, W. J., Sachs, P., Sierhuis, M., & van Hoof, R. 1998. Brahms: Simulating practice for work systems design. *Int. J. Human-Computer Studies*, *49*, 831-865.

Clancey, W. J. 2002. "Simulating activities: Relating motives, deliberation, and attentive coordination," *Cognitive Systems Research*, *3*, No. 3, 471-499.

Clancey, W. J. 2003. Principles for integrating Mars analog science, operations, and technology research. *Workshop on Analog Sites and Facilities for the Human Exploration of the Moon and Mars,* May 21-23, Colorado School of Mines, Golden, CO.

Clancey, W. J. 2004. Automating Capcom: Pragmatic operations and technology research for human exploration of Mars. In C. Cockell (ed.) *Martian Expedition Planning,* Vol. 107, AAS Science and Technology Series, pp. 411-430.

Clancey, W. J. 2004. Roles for agent assistants in field science: Understanding personal projects and collaboration. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews,* 34 (2) 125-137. Special Issue on Human-Robot Interaction, May.

Clancey, W. J. and Lowry, M. 2012. Lunar surface systems study: Interoperability. NASA Technical Publication 2012-216041.

Clancey, W. J., Sierhuis, M., Alena, R., Berrios, D., Dowding, J., Graham, J.S., Tyree, K.S., Hirsh, R. L., Garry, W.B., Semple, A., Buckingham Shum, S.J., Shadbolt, N. and Rupert, S. 2005. Automating CapCom using Mobile Agents and robotic assistants. NASA Technical Publication 2007-214554. Washington, D.C. Available: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20070035904_2007036018.pdf

Clancey, W. J., Sierhuis, M., Alena, R., Dowding, J., Scott, M., van Hoof, R. 2006. Power system agents: The Mobile Agents 2006 field test at MDRS. *Mars Society Annual Convention*. Available:

http://homepage.mac.com/wjclancey/%7EWJClancey/ClanceyMarsSoc2006.pdf

Clancey, W. J., Sierhuis, M., Dowding, J., Berrios, D., Scott, M., van Hoof, R., Delgado, F., Tourney, S., & Kosmo, J. 2007. Mobile Agents integrate astronauts, rover, and mission support In Desert-RATS mission simulation. *Mars Society Annual Convention* (abstract). Los Angeles.

Clancey, W.J., Sierhuis, M., Seah, C., Buckley, C., Reynolds, F., Hall, T., & Scott, M. 2008. Multi-agent simulation to implementation: a practical engineering methodology for designing space flight operations. In *Engineering Societies in the Agents' World VIII*. Lecture Notes in Artificial Intelligence (Artikis, A., O'Hare, G., Stathis, K., & Vouros, G., Eds.), Vol. 4995, pp. 108–123. Heidelberg: Springer.

Clancey, W.J., Sierhuis, M., Nado, R., van Hoof, R. 2010. Collaborative infrastructure conceptual overview. TM10-0001 NASA Ames Research Center, unpublished.

Clancey, W. J., Lowry, M., Nado, R., Sierhuis, M. 2011. Software productivity of field experiments using the Mobile Agents open architecture with workflow interoperability, *IEEE Space Mission Challenges for Information Technology*, August 2011, Palo Alto, pp. 85-92.

Dowding, J., Alena, R., Clancey, W. J., Graham, J., and Sierhuis, M. 2006. Are you talking to Me? Dialogue systems supporting mixed teams of humans and robots. *AAAI Fall Symposium 2006: Aurally Informed Performance: Integrating Machine* Listening and Auditory Presentation in Robotic Systems, October, Washington, DC.

Hirsh, R., Graham, J., Tyree, K., Sierhuis, M., Clancey, W. J. 2006. Intelligence for human-assistant planetary surface robots. In A. M. Howard and E. W. Tunstel (Eds.) *Intelligence for Space Robotics***,** pp. 261-279. Albuquerque: TSI Press.

Johnson, A. W., Newman, D. J., Waldie, J. M., Hoffman, J. A. 2009. An EVA mission planning tool based on metabolic cost optimization, SAE 2009-01-2562, *39th International Conference on Environmental Systems*, Savannah, GA, 12-16 July 2009.

Johnson, A.W., *An Integrated EVA Mission Planner for Future Planetary Exploration*, M.S. thesis, Massachusetts Institute of Technology, 2010 (forthcoming).

Kaskiris, C., Sierhuis, M., Clancey, W. J., van Hoof, R. 2005. Mobile Agents: A ubiquitous multi-agent system for human-robotic planetary exploration. *2nd International Symposium on Systems & Human Science*, San Francisco.

Pedersen, L., Clancey, W. J., Sierhuis, M., Muscettola, N., Smith, D.E., Lees, D., Rajan, K., Ramakrishnan, S., Tompkins, P., Vera, A., Dayton, T. 2006. Field demonstration of surface human-robotic exploration activity. *AAAI-06 Spring Symposium: Where no human-robot team has gone before*.

Rader, S. 2008. Constellation's Command, Control, Communications, and Information Architecture (C3I) Overview. Software & Avionics Integration Office (SAVIO) PowerPoint presentation, December 11.

Sierhuis, M. 2001. *Modeling and simulating work practice.* Ph.D. thesis, Social Science and Informatics (SWI), University of Amsterdam, The Netherlands.

Wooldridge, M. 2002. *An Introduction to MultiAgent Systems.* Chichester, UK: John Wiley & Sons Ltd.

Appendix I. Mobile Agents Software Architecture Overview

This appendix provides more detail about the Mobile Agents Architecture. Workflow software agents in the Mobile Agents field experiments were implemented in the Brahms Language (Clancey et al., 1998; Sierhuis, 2001). To employ a common messaging scheme each subsystem to be integrated is "agentified"—it is made to behave like a Brahms workflow agent by "wrapping" it in a Brahms Communications Agent using the Brahms JAVA API.

A subsystem is made into an agent (i.e., integrated) by defining SpeechActs (structured messages) by which it will interact with other agents in the system. These SpeechActs must implement any overarching defined protocol for communication among agents. Thus, the API of the subsystem is extended by the Communication Agent to translate its API calls to SpeechActs and vice versa. The subsystem API and Communication Agent therefore work together (acting as the wrapping) to enable communication with any workflow agent in the system:

{Workflow Agents} ⇔ Communication Agent ⇔ Subsystem API ⇔ Subsystem

Other agent-based languages, such as JADE with its FIPA-like structure using SpeechActs, could be used instead of Brahms to implement the architecture described here. However, Brahms effectively provides direct support for an open architecture with interoperability via its SpeechAct and Communication Agent (CA) structures. Recent work on service-oriented architectures (SOA) promotes a similar common messaging approach for interoperability, in which subsystems provide "services."[9] The MAA illustrates how to provide an SOA by converting subsystems into agents. The variety of MAA configurations described in this study support the claim that agents are a good way to implement a software service; in particular, to make a component into a service "agentify" it.

The MAA provides an open architecture with interoperability by extending CORBA in a way that raises it from the level of direct functional communications to the level of services. In doing this, the MAA shifts system design from the level of *software objects* and processes to the level of *agents* who communicate using the language of components and operations in which people naturally describe their goals and activities (e.g., "Scout, take a picture of Astronaut 2"; "Extend the duration of Surveying Worksite 2 by 10 minutes"). Here we provide a brief overview of how MAA extends CORBA.

In itself, CORBA only enables exposing internal system classes in one object-oriented program to another program. This means that a program that uses exposed CORBA classes needs to understand the internal functions of the system with which it interacts. Every system has its own internal structure and functions, perhaps written in different programming languages. For example, CORBA enables calling C++ object methods from

[9] Mishkin's backup slides "Command & Control" in Rader (2008) characterize "services" as how C3I subsystems are "providing the endpoints of data flows" (p. 57). The term "services" appears throughout the roadmap in reference to network, security, and common C2 functions.

Java and vice versa. So, provides language independence and also allows calling methods in distributed systems with a delayed response (the notion of a "callback function"). CORBA enables inter-process communication, by allowing indirect access to the local storage of other processes.

In contrast with systems using CORBA and other agent-based middleware architectures (for examples, see Bridges and Vladimirova [2011]), the composite APIs (communication agents) approach does not expose a software process' internal methods/ functions. It is based instead on a more abstract communication protocol that defines how agents communicate messages to each other (their structure and language) and requires agents determine how to respond to these messages (by providing data or causing actions to occur). The benefit of the communication agent approach that programmers do not need to provide CORBA object definitions to each developer of the integrated system and which each developer must use. Instead, the team defines a communication message protocol— in MAA this is the syntax and language of SpeechActs—and then the developers may work independently.

Other systems use agents as a form of middleware to coordination operations, such as a distributed satellite system (Bridges and Vladimirova 2011). The use of communication agents in the MAA provides an open architecture, specifically facilitating interoperation of legacy hardware and software systems of any type (e.g., contrast with the very different problem of integrating a network of identical components designed for formation flying or a sensor network). Consequently, the MAA would be a candidate architecture for developing certain kinds of "virtual satellites"—"a spatially distributed group of satellites working as a single unit to perform a specific mission" (Bridges and Vladimirova 2011), where the component satellites provide different services and/or include legacy systems.

In summary, Brahms provides a language for creating agents and a communication infrastructure for agents to interact. Actual transmission of messages occurs within and through the Brahms Virtual Machine, the agent executive that runs on each platform containing agents. In the MAA implementation used in the systems described in this report, a system called KaOS with its central directory service enabled agents to communicate across distributed platforms (e.g., see **Figure 13** and **Figure 14**). KaOS used CORBA to implement the actual message transmissions. Because agents communicate using SpeechActs, system developers did not need to know about CORBA. For example, the ERA's API exposed its C++ methods using CORBA objects which the ERA CA, written in Java, translated to and from SpeechActs used by agents throughout the exploration system. A later generalization, called the Collaborative Infrastructure (CI) used in ETDP A40 ("autonomy for operations") projects and in OCAMS in ISS Mission Control, handles this translation by providing a toolkit of C++ and Java libraries that include SpeechActs for "agentifying" an external system (Clancey, et al. 2010).

Conventionally, an object has only internal properties and attributes. In contrast, agents can have beliefs about other agents, enabling them to model the state of subsystems as well as the world of people and the environment. Also, in the MAA agents are inherently

goal-directed through the Plan Assistants, which maintain an open task structure; object-oriented design doesn't in itself provide this functionality. In practice, every SpeechAct is a task that the agent needs to execute; it is deleted when a response is returned. By periodically polling and acting on open tasks and agent may select an alternative approach for accomplishing a task as well as provide a warning to the requester that there has been an delay (e.g., "Scout is not responding to your request to take a picture").

In summary, the MAA enables arbitrary object-oriented systems to become agents not just by exposing their methods, but rather by wrapping the software so communications between subsystems occur using task-level messages (SpeechActs). The Communication Agent (CA) is the Java program that straddles the programming domain of the subsystem (its language, data, and functions) and the agent domain of the integrated workflow system (represented in terms of the objects and activities of the EVA system). Using a common language to integrate data and functionality follows the principles of "model-based" programming, which enables relating semantically different data across hardware and software systems (Kaskiris, et al. 2005). Consequently, designing an agent to command a robot is handled in the same manner (at the same semantic level) as designing an agent to associate a photograph with a map location and sample bag, based on the voice command of an astronaut. This commanding language among agents and subsystems is derived on a theory of "speech acts," which in MAA follows the FIPA protocol.

Using the MAA, components with appropriate APIs for means of communicating can be configured into workflow systems, in which data, command, and information is translated, transformed, interpreted, and conveyed to support the work people are doing. For example, data flowing from sensors to monitoring software is directed by alerting functions to output media so it is accessible within the work context (e.g., on a heads-up display or on a voice loop). Figure 12 provides a broad overview of the hybrid communications in the overall EVA exploration systems developed using the MAA. Details are provided in subsequent sections, explaining in particular how Communications Agents were created and used to produce a comprehensive workflow system in a variety of contexts.

PEOPLE	Microphone & Headphone	PERSONAL AGENTS	Comms Agents	EXTERNAL SYSTEMS
Astronaut1	←voice→	**Agent$_{Astro1}$**	←API→	GPS, ERA Biosensors Camera
| radio |		| wireless network |		
HabCom	←voice→	**Agent$_{HabCom}$**	←API→	Email (to RST)
| radio |		| wireless network |		
Astronaut2	←voice→	**Agent$_{Astro2}$**	←API→	GPS, ERA Biosensors Camera

Figure 12. Mobile Agents Architecture Communications Schematic Relating People, Agents, and External Systems. Voice commanding involves a hybrid of methods for communicating, including spoken voice (microphone and headphone), radio, wireless network, and software applications interfaces. Key: RST = Remote Science Team; ERA = EVA Robotic Assistant. External systems are illustrative. Communication Agents use APIs of external systems to interface with workflow agents.

Appendix II. Field System Configuration Diagrams

This appendix provides a chronological sequence of configuration diagrams for the key exploration systems used in the Mobile Agents field experiments (2002-2006). See references for details.

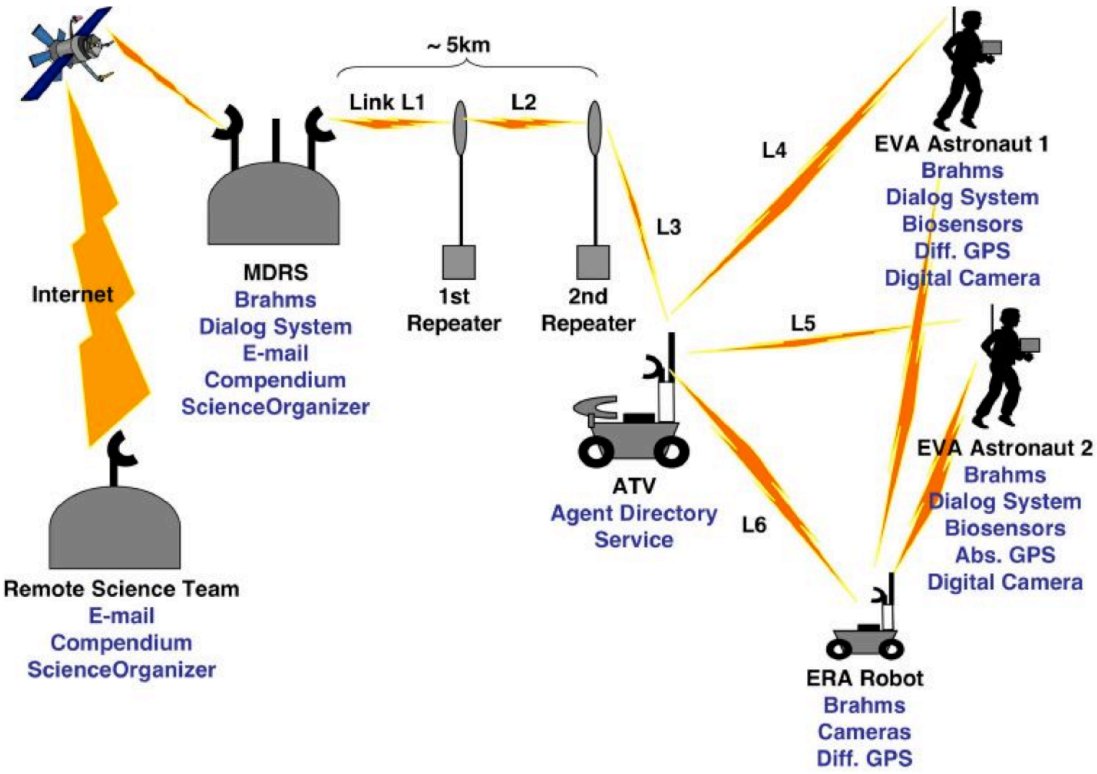

Figure 13. Wireless Network Configuration for MDRS04 for Hardware and Software Components. Centralized directory service (implemented in KaOS) enabled agents to locate each other for requesting and providing data and commanding integrated subsystems. MDRS05 configuration was similar with a second ERA Robot. Agent details appear in
Figure 14.

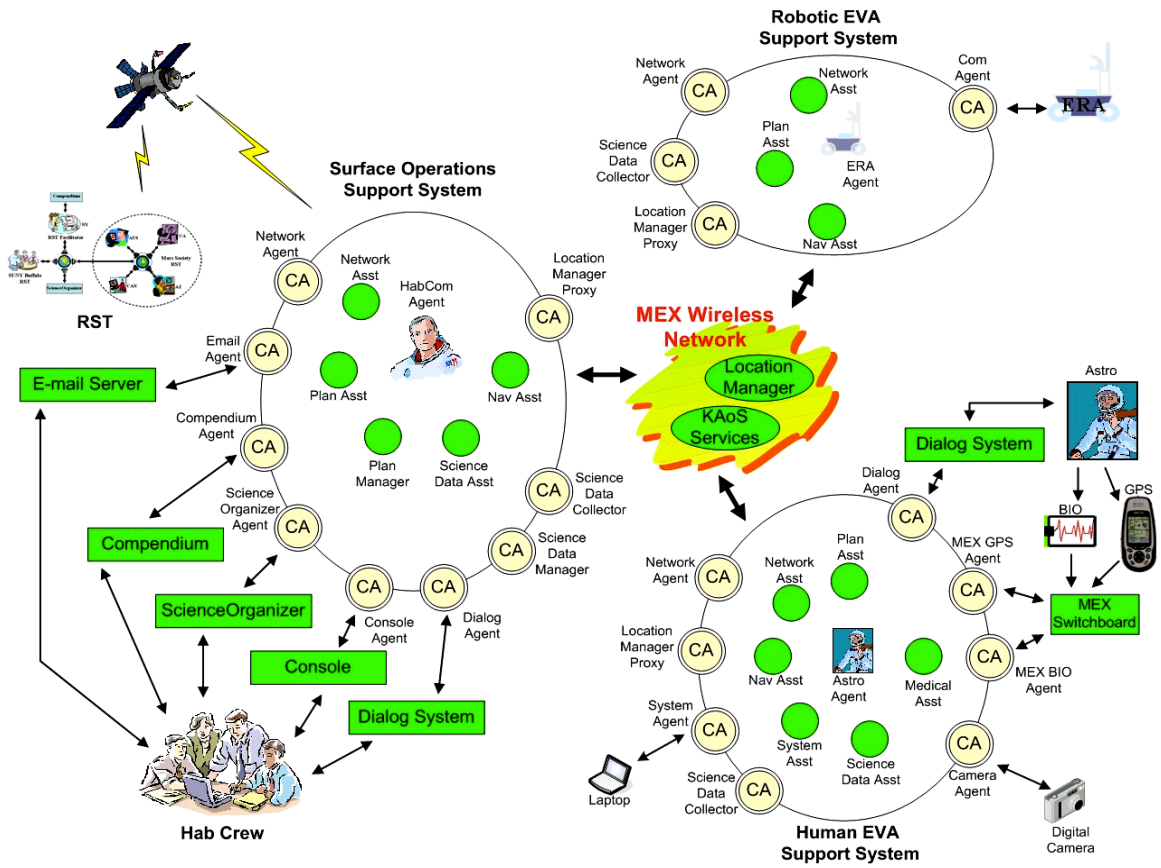

Figure 14. MDRS05 EVA Exploration System Configuration. The EVA system included two astronaut systems and two Robotic EVA systems (ERAs) as shown. Remote Science Team was distributed in USA, Australia, and England (Clancey, et al., 2005; Hirsh, et al. 2006). This configuration is the mature version of the four year sequence: the general structure was introduced in DRATS02, distributed platforms introduced in MDRS03, and the remote science team added for MDRS04. CDS05 and DRATS05, which followed four months later, changed the robots and added additional external software and platforms (see following figures). The green circles represent agents that constitute the workflow backbone of the exploration system.

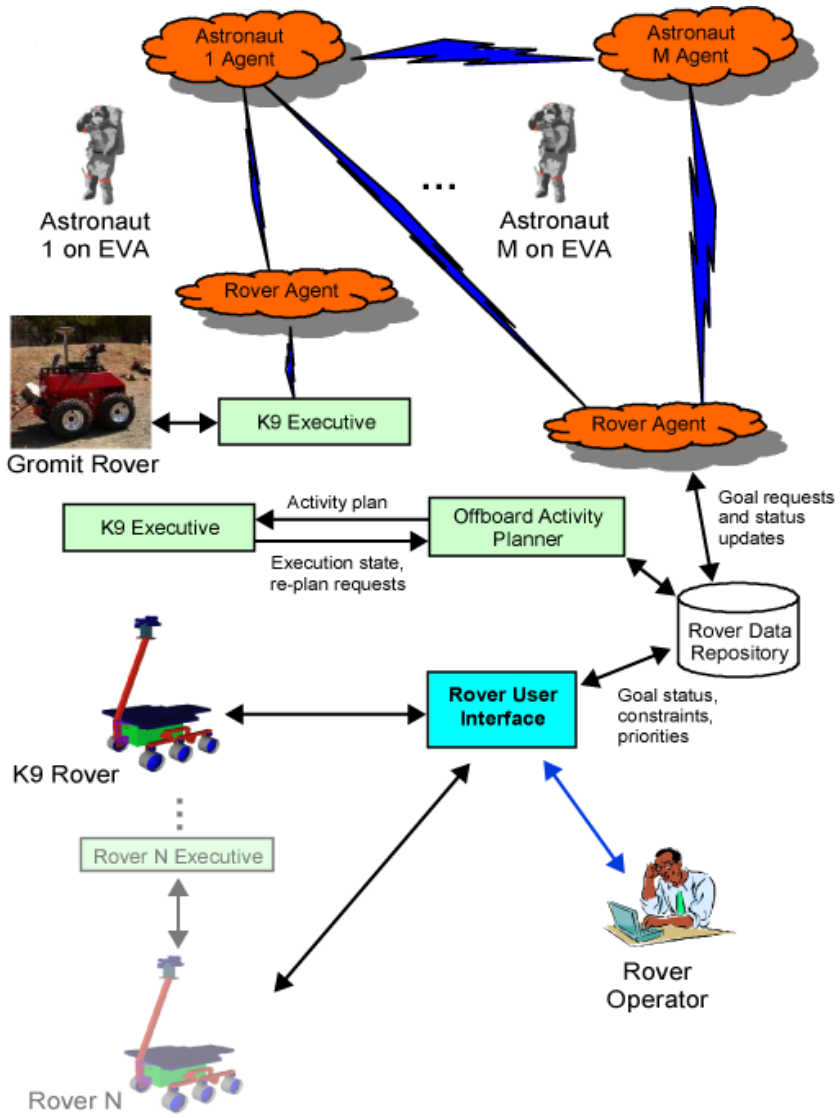

Figure 15. CDS05 EVA Exploration System Configuration. The field test involved one astronaut and one K9 Rover; however, the architecture allowed EVA configurations with multiple astronauts and rovers *without changing the software*. Goal-oriented commanding of the K9 robot is enabled for both the astronaut (via verbal requests mediated by agents, e.g., "Inspect rock named Broccoli when able") and the rover operator in the habitat (via direct manipulation of the visual display interface). Prepared plans initiated by astronauts on EVA did not require rover operator intervention (Pedersen, et al., 2006).

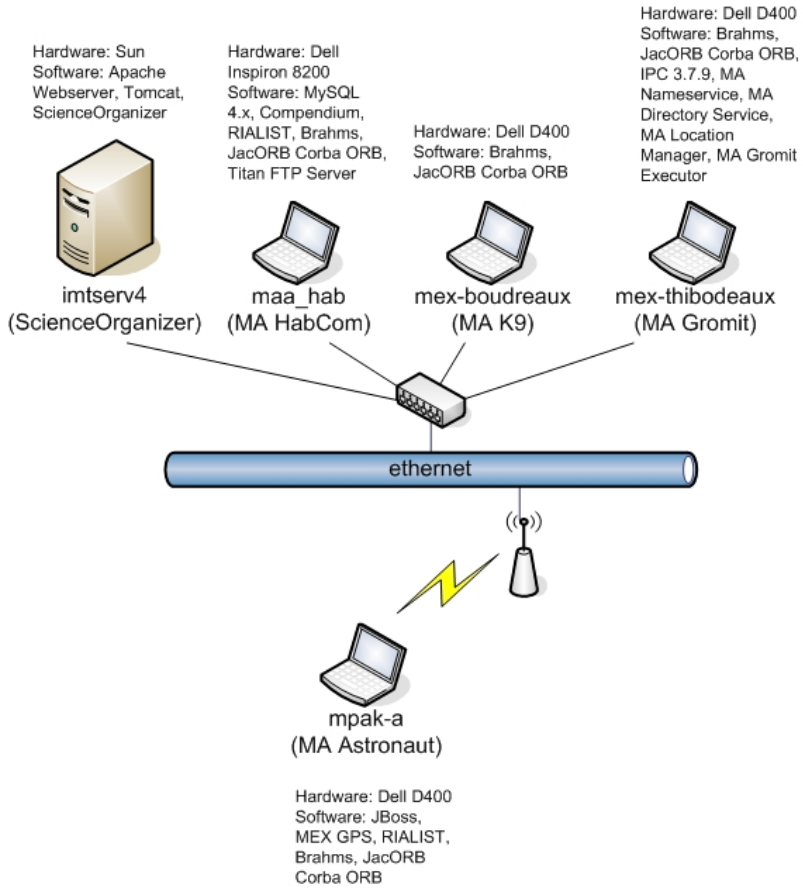

Figure 16. CDS05 Platform and Network Configuration. K9 and Gromit personal agents are directly adapted from those used in the MDRS05 configuration (ERAs named Boudreaux and Thibodeaux). (Slide provided by Rick Alena, NASA/Ames.)

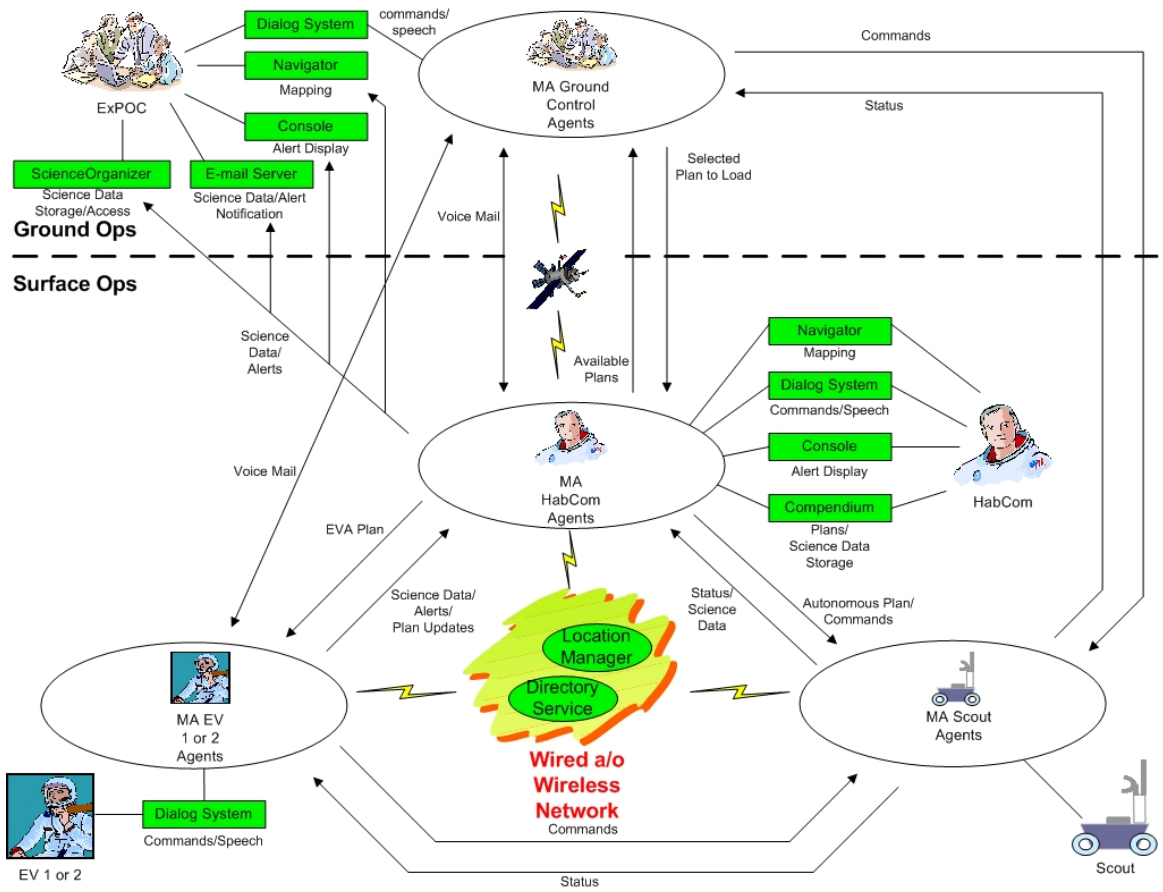

Figure 17. Desert-RATS 2006 EVA Exploration System Configuration (Meteor Crater, September 2006; Clancey et al. [2007]). Reconfiguration of MDRS05 to use Scout rover instead of ERAs, using agents to control its Geophone deployment system from the ExPOC agent platform in Houston. This configuration demonstrated how two kernel support systems, one local (HabCom) and the other remote (ExPOC), could be used to coordinate the flow of data, information, and goal-directed commands coming from EVA astronauts, the surface habitat operator, and the remote ground support operator.

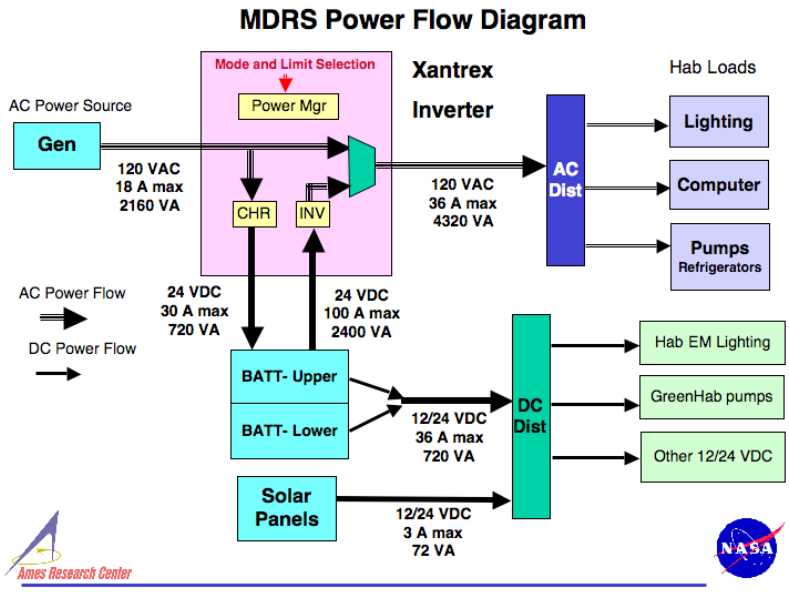

Figure 18. Power System Configuration of Mars Desert Research Station (April 2006). Xantrex inverter contained a power management system for charging or drawing from habitat backup battery system, receiving AC input from the diesel generator. DC power was also provided by solar electric panels. The OneMeter Channel Meter system instrumented these various sources to provide data to the Power Agent System (**Figure 19**).

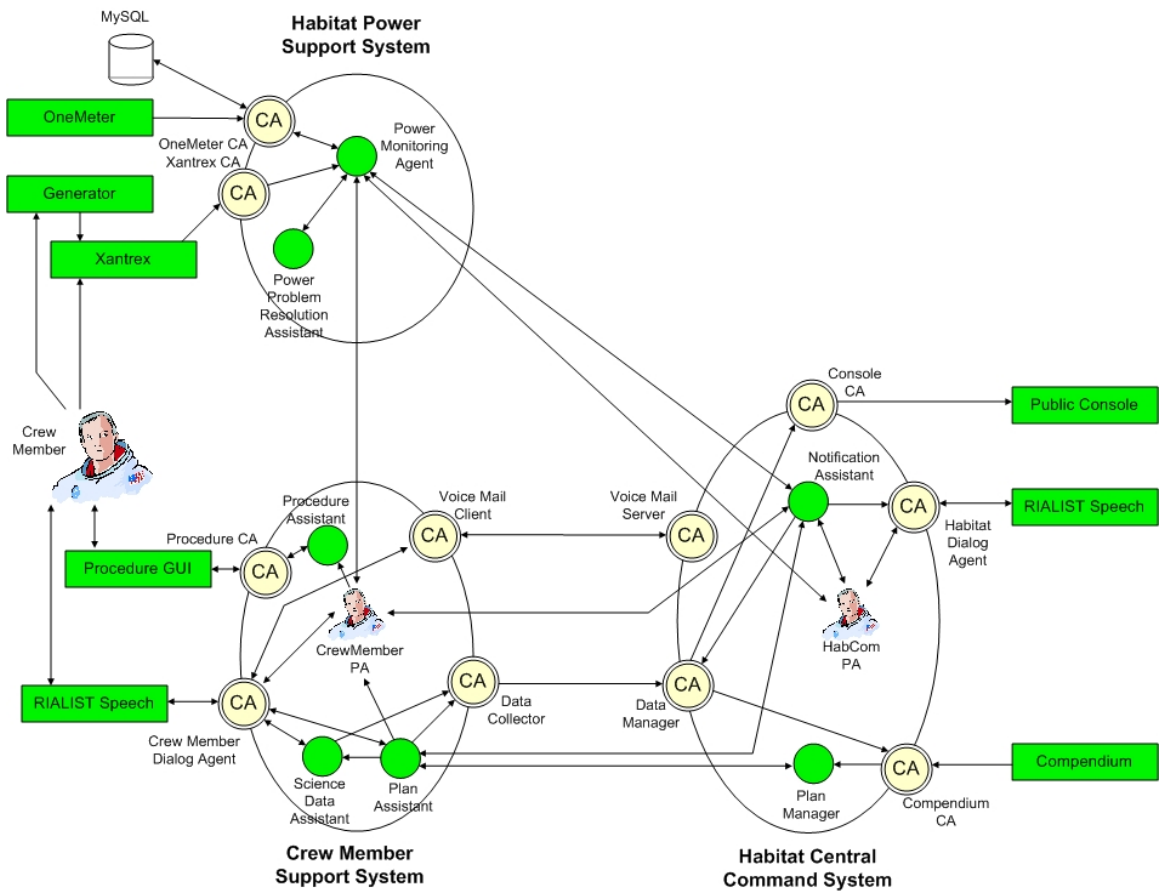

Figure 19. Mobile Agents Power Agents Configuration. The crew and HabCom systems include a "personal agent" for coordinating communications (command processing, alerting, and dataflow) with crew members. The link to the Xantrex inverter (**Figure 18**) provided setup information (e.g., error thresholds); real-time data came only from the OneMeter system. (Graphic by van Hoof). In the tested configuration, four copies of the Crew Member Support System were running on four different laptops, initialized by the first name of the crew member. HabCom's Voice Mail Server CA receives voice mails from any crewmember and transmits them to the requested crewmember (e.g., "Send a voice note to Bill at 1130 AM…."). See text below for elaboration.

The power system in the Mars Desert Research Station during April 2006 consisted of an external generator, batteries under the habitat, solar panels, and inverter. A previous crew had previously deployed the OneMeter (Brand Electronics) electric metering system to instrument the various power sources. The Mobile Agents configuration consisted of six laptop computers. One networked computer (HabCom) was placed on the upper deck, connected to loudspeakers. Four other laptops were paired with wireless (Bluetooth) headsets that enabled crew members to interact with their Personal Agents from anywhere in the hab (plus in one test as far as 10 meters outside). A sixth laptop functioned as a telenet server for the OneMeter device, constituting the Power Support System (for more details and graphics, see Clancey et al., 2006).

Referring to Figure 19, the Power Support System contains five new agents developed for this configuration and setting:

1. Power Monitoring Agent—processes incoming data from the OneMeter system, as well as answers queries about historical data.
2. Power Problem Resolution Assistant—Receives an error condition event from the Power Monitoring Agent; determines the procedure to be followed to resolve the problem; and sends that procedure back to the Power Monitoring Agent.
3. Procedure Assistant—Receives procedure relating to a fault mode for display to HabCom operator.
4. One Meter Communication Agent (OneMeter CA)—serves as a telnet server. The OneMeter connects to this telnet server every 15 minutes (a fixed property of the OneMeter firmware) and sends the data for all channels. The agent parses and packages the data and sends it to the Power Monitoring Agent. The data are also stored in a MySQL database, which is used for responding to historical queries.
5. Xantrex Communication Agent (Xantrex CA)—on request, provides parameter settings for the Xantrex Inverter to the Power Monitoring Agent.
6. Procedure GUI Communication Agent (Procedure CA)—interacts with the HabCom GUI to display information relating to power problems, communicated by the Procedure Assistant.

The key work is performed by the Power Monitoring Agent:

- *Incoming Data:* On every new reading received from the OneMeter Communication Agent, evaluates the values for the channels using a set of rules. The agent is designed to detect five anomalous events (e.g., impending shut-down of inverter due to low battery voltage). If an anomalous event is found, the agent sends the condition data to the Problem Resolution Agent for analysis. It forwards the alert with the procedure to the Notification Assistant in the Habitat Central Command System (HabCom laptop), which distributes the alert and procedure to the Personal Agents of the crew and HabCom systems that have subscribed to this alert. The procedure is extracted by these Personal Agents and sent to their respective Procedure Assistants, which display the procedure on the associated display. The alert message is also sent to the respective Dialog Agent for verbal notification.
- *Historical Queries:* On receiving a power system inquiry from a crew member or HabCom Personal Agent, sends the query to the OneMeter Communication Agent, and returns the desired data (e.g., "what was the maximum habitat amperage since yesterday?").

The MDRS49 configuration also included the activity plan loading and tracking functionality used for previous EVA simulations. In particular, crew members had daily schedules that indicated what activity they would normally be doing at a given time. Crew members could inquire about anyone's current activity. This was part of a related investigation of how a capability such as the Power Monitoring System might be augmented by information about the crew's activities.

As an example of repurposing existing agents, an ability to send voice mail among crew members was designed and implemented during the two-week field experiment. The voice message is recorded as a standard "voice note," which is sent to the science data assistant, from which it is passed to the data collector, the data manager, and then to the Compendium database (this illustrates how communication agents may interact directly, without the mediation of workflow agents). When a voice mail is sent out the RIALIST CA (dialog agent) also sends the voice note to the voice mail client, from which it is sent to the voice mail server and then to intended recipient at the scheduled time, according to the sender's request.

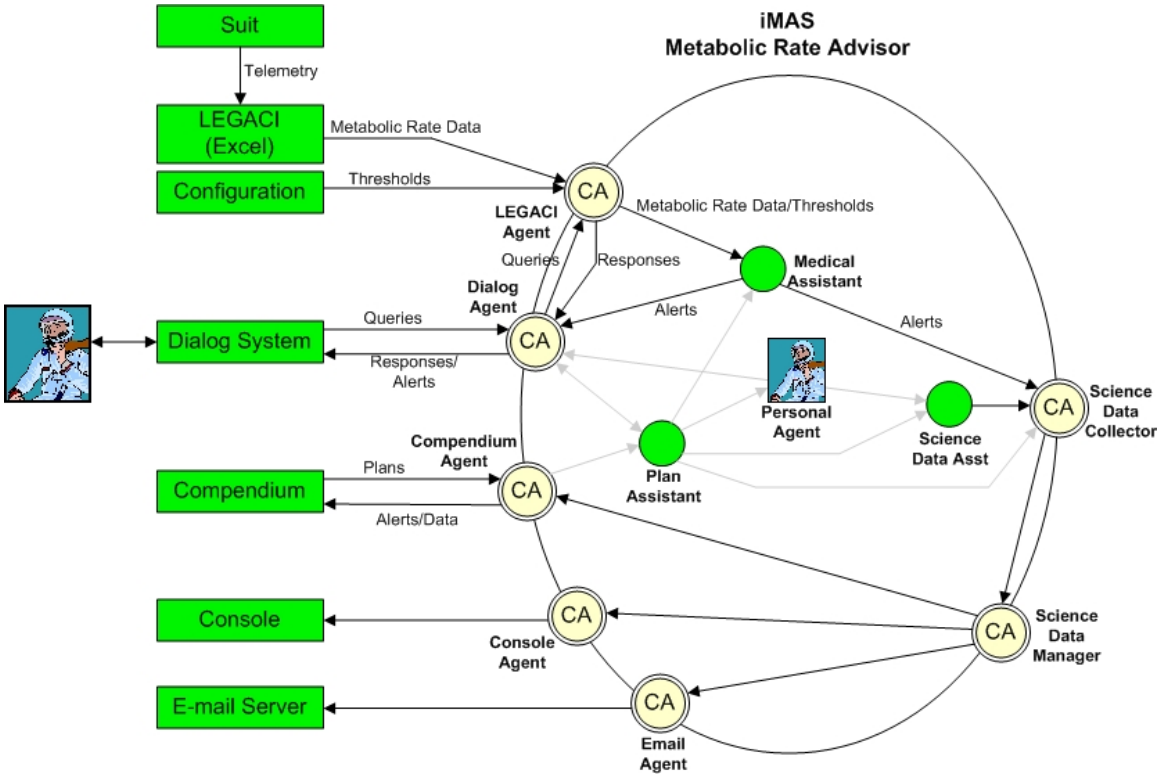

Figure 20. Metabolic Rate Advisor (POGO, a variant of iMAS, the Individual Mobile Agent System). The Mobile Agent system referred to as POGO07 is a variant of the standalone Mobile Agents system, for use on a single platform that during operation is not necessarily communicating with other agent platforms. Two agents were added: the LEGACI CA, which received metabolic rate and consumables data from the LEGACI program implemented in Excel, and the Medical Assistant workflow agent, which interpreted the data, transmitted alerts to the crew member (via the Dialog Agent), and responded to the crew members requests for status information. LEGACI received telemetry data from the pressurized spacesuit life support system and biosensors worn by the astronaut in the partial gravity simulator (POGO) at Johnson Space Center.

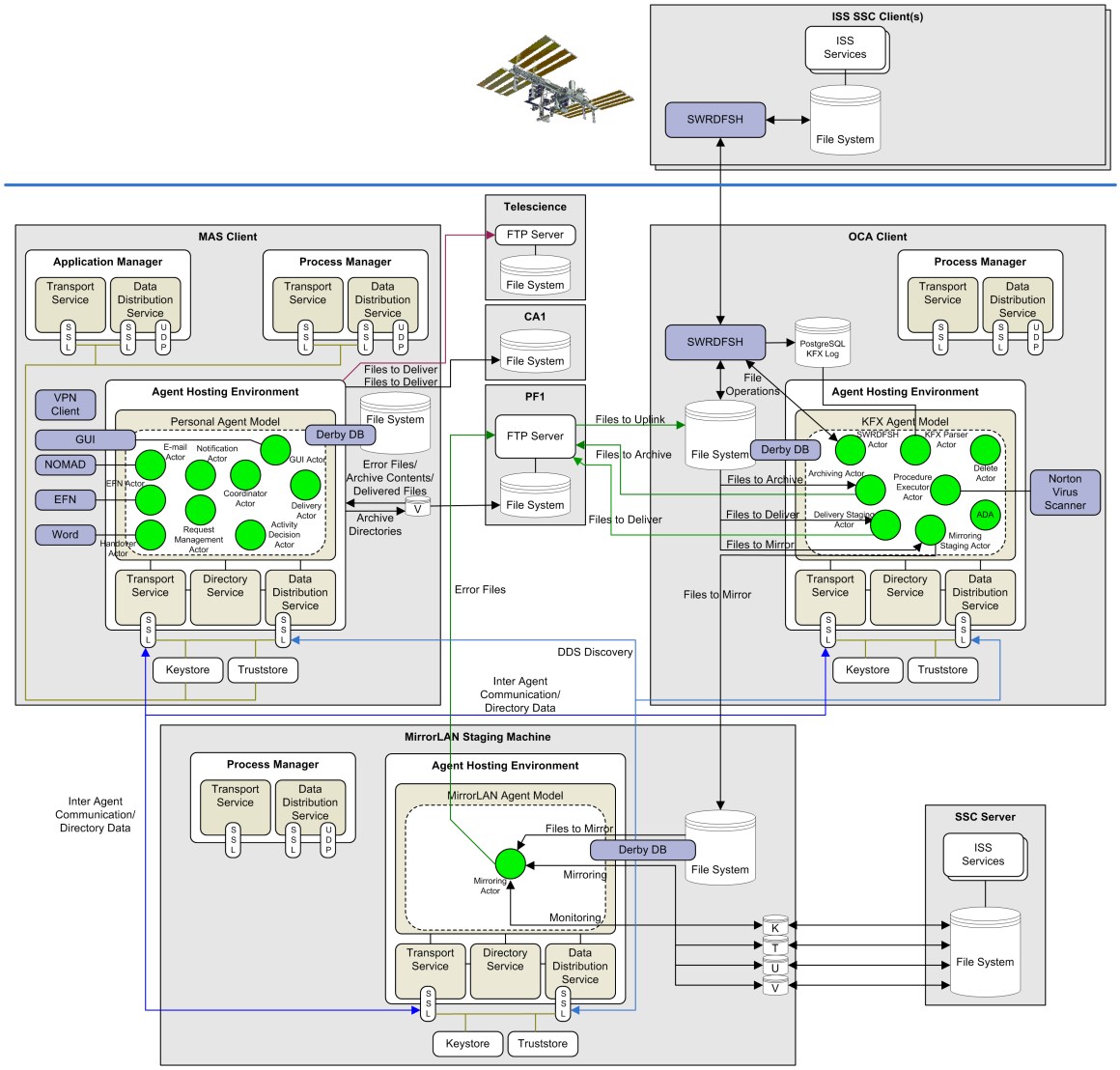

Figure 21. Orbital Communications Adapter (OCA) Management System (OCAMS), Revision 4 Workflow System for ISS File Communications. Green circles are agents; Blue rectangles are components (e.g., EFN flight note system, Microsoft Word, NOMAD email, SWRDFSH FTP). R3 deployed in early 2010 automates mirroring, archiving, logging, delivery and notification of files transferred between ISS crew and ground support. R4 will automate uplink and downlink using SWRDFSH; ground support teams will make requests to OCAMS through flight notes and email; files are usually transferred using drop-boxes (dedicated folders in JSC MCC and onboard the ISS). Estimated 80% of previous 24-7 MCC Backroom Officer position is automated. This architecture uses the *Collaborative Infrastructure* for data transfer across multiple network with different security systems (see glossary). In a typical configuration, agents are networked over two MAS (flight controller) platforms, two OCA clients networked to the ISS, and the MirrorLAN staging machine for creating the mirrored file system.

Appendix III. Information and Goal-Oriented Services Provided by Workflow Agents in Field Experiments

This table lists the hardware and software components that were integrated in the indicated system configuration, followed by the workflow capabilities. A plus ("+") indicates that the component or capability was introduced for that configuration; an X indicates that it was included.

	DRATS02	MDRS03	MDRS04	MDRS05	DRATS05	CDS05	PA06	iMAS06	DRATS06	POGO07	iMAS08
Hardware Integration											
EVA Agent Platform (used by Astronaut)	+										
Stanford Biovest medical sensors	+	X									
Boudreaux (ERA) Robot	+	X	X	X							
Additional Astronaut Platforms (2 laptops)		+	X	X	X				X		
Habitat Monitoring & Server Platform		+	X	X	X	X	X		X	X	
Digital Camera(s)		+	X	X	X	X		X	X		X
Nonin biosensors			+	X	X	X		X	X		
Laptop hardware sensors			+	X	X	X	X	X	X	X	X
Thibodeaux (ERA) Robot				+							
Panoramic Camera (on robot)				+	X				X		
Video Camera (on robot)				+	X						
CAI backpack (JSC, Glenn)					+				X		
Heads Up Display (Hamilton Sundstrand)					+				X		
SCOUT Rover (JSC)					+				X		
K9 Robot (Ames/Fong & Pederson)						+					
Gromit Robot (Ames/Muscettola)						+					
Bluetooth headsets worn by habitat crew							+				
Electric power monitors (OneMeter)							+				
Solar Electric system							+				
DC Battery system							+				
Electric DC-AC inverter system							+				
Additional Astronaut Platforms (4 laptops)							+				
Geophone Deployment Device									+		
Remote monitoring & control platform (ExPOC)									+		
Space suit audio system									+	X	
Suit life support sensors (e.g., consumables)										+	
SCUBA headset											+
Software Integration											
RIALIST (voice commanding)	+	X	X	X	X	X	X	X	X	X	X
Email System		+	X	X	X	X			X		
Compendium Database			+	X	X	X	X	X	X	X	X
Science Organizer Web Interface			+	X	X	X			X		
Europa Planner (Ames)						+					
Metabolic Rate Advisor (Legaci in Excel/VBA)										+	

	DRATS02	MDRS03	MDRS04	MDRS05	DRATS05	CDS05	PA06	iMAS06	DRATS06	POGO07	iMAS08
Astronaut Health Monitoring											
Log Biosensors (heart, temp, SPO2)		+	X	X	X	X		X	X	X	
Log biosensors "every N seconds/minutes"			+	X	X	X		X	X	X	
Voice alerts about heart rate, temperature, SPO2			+	X	X	X		X	X	X	
Change thresholds for SPO2 & heart rate alerts				+							
Voice queries about met rate, k-cal usage, body heat storage, sweat loss										+	
Alerts about met rate, k-cal, heat, sweat										+	
System Health Monitoring											
Alert for Laptop low battery			+	X	X	X	X	X	X	X	X
Alerts of network failures affecting command processing			+	X	X	X	X		X	X	X
Alert for Laptop CPU temperature too warm				+	X	X	X	X	X	X	X
Habitat Batteries status (voltage, amps, charging), e.g., "Are the batteries charging?"							+				
Generator status (voltage, amps, power allocation to habitat and batteries)							+				
Solar panel power generation status (amps)							+				
Habitat power status (voltage and amps usage) {now | <at time in past>}							+				
Inverter status (Supplying power, source)							+				
"When did the generator go offline/online?"							+				
"When did the batteries start dis/charging?"							+				
"What did the generator/batteries' status change?"							+				
Alert when generator is offline							+				
Alert when habitat amp use exceeds nominal							+				
"What was the {Max|Min|Avg|Value} {voltage|power| of {habitat|generator|batteries|solar} {since|during} {time in past | period}?"							+				
"How many hours will batteries last at current draw?"							+				
Automatic display of repair procedure on system error alert							+				
Ask for parameter setting (e.g., "What is low batter cut out voltage?"							+				
Ask about life support power status										+	
Ask about life support consumables status										+	
Ask about suit 02 leakage										+	
Ask about life support scrubber, feedwater, inlet temperature status										+	

	DRATS02	MDRS03	MDRS04	MDRS05	DRATS05	CDS05	PA06	iMAS06	DRATS06	POGO07	iMAS08
Alerts about life support scrubber, feedwater, inlet temperature status										+	
Alerts about power, consumables, suit 02										+	

Location Tracking	DRATS02	MDRS03	MDRS04	MDRS05	DRATS05	CDS05	PA06	iMAS06	DRATS06	POGO07	iMAS08
GPS location logging of astronaut and robot	+	X	X	X	X	X		X	X	X	X
"Where am I?" (GPS coordinates)		+	X	X	X	X		X	X	X	X
"Where is <location>?" (GPS coordinates)		+	X	X	X	X			X		
Location information sent to Remote Science Team (RST) at variable intervals		+	X	X	X	X			X		
Location naming by astronaut (waypoint # or from predefined list of nouns)		+	X	X	X	X		X	X	X	X
GPS location logging of robots on demand			+								
Alert when lose GPS tracking			+								
Global naming management to avoid duplicate definition			+	X	X	X		X	X	X	X
Location names include predefined map locations (e.g., Lith Canyon)				+	X	X		X	X	X	X
Distinction among Workstation, Worksite, and Waypoints				+	X	X		X	X	X	X
Query relative locations of astronauts, robots, & habitat for navigation assistance (bearing & distance)				+	X	X		X	X	X	X
Allow references to "here" "X's current location"				+	X	X			X		
Dynamic map location on head-up display					+	X			X		

Human-Robot Coordination	DRATS02	MDRS03	MDRS04	MDRS05	DRATS05	CDS05	PA06	iMAS06	DRATS06	POGO07	iMAS08
Ask robot to follow/stop following you		+	X	X	X	X			X		
Ask robot to halt		+	X	X	X	X			X		
"Take a picture of {me | <location name>"		+	X	X	X	X			X		
"<robot> go to <location>" (e.g., waypont #, "come here" "return to habitat")		+	X	X	X	X			X		
Ask robot to go to waypoint and wait N minutes		+	X	X	X	X			X		
"Where is <robot>?"		+	X	X	X	X			X		
"Take a panorama image {at <location>}"			+	X	X				X		
Ask robot to execute a movement plan			+	X	X	X			X		
Ask robot to watch <astronaut> (video camera)			+	X	X	X			X		
Ask robot to send its traverse map to HabCom			+	X	X	X			X		
Ask robot to repeat its current activity			+	X	X	X			X		

	DRATS02	MDRS03	MDRS04	MDRS05	DRATS05	CDS05	PA06	iMAS06	DRATS06	POGO07	iMAS08
Ask robot to shift to its next activity			+	X	X	X			X		
Ask robot if it has network connectivity {to <astronaut> | <robot>}				+							
Command robot to enable/disable network relay				+							
Ask robot to enable/disable voice commanding				+	X	X			X		
Ask robot whom it is following				+	X	X			X		
Ask robot whom it is watching				+	X	X			X		
Ask robot to power on/off <subsystem> (e.g., brakes, headlights, motors)				+	X				X		
Answer robot by "Yes/No, <robot name>"				+							
Robot interprets "there" to be last location reference				+	X	X			X		
Robot interprets "to <astronaut>" to mean that person's current location				+	X	X			X		
Tell robot to define its current location as <waypoint #>				+	X	X			X		
Tell robot to create an elevation/ground track map				+							
Ask robot to inspect an area/waypoint {using <instrument>}						+					
Tell robot to move N <units> forward/backward						+			X		
Tell robot to turn N degrees left/right						+					
Ask robot to describe its status						+					
Ask robot to plan a path to <location>									+		
Ask robot to prepare to follow astronaut									+		
Ask robot to watch <location>									+		
Ask robot to execute instrument procedure									+		
Plan Management											
Activity tracking (time & loc)		+	X	X	X	X	X	X	X	X	X
Alert for deviation from planned activity duration		+	X	X	X	X	X	X	X	X	X
Alert for deviations from planned activity location		+	X	X	X	X	X	X	X	X	X
Alert about total EVA time remaining		+									
Voice command to start selected/"next" activity		+	X	X	X	X	X	X	X	X	X
Change Activity duration or distance thresholds		+	X	X	X	X	X	X	X	X	X
"How much time is left (for current activity)?"		+	X	X	X	X	X	X	X	X	X
Start selected activity at <location name> {for <duration>}			+	X	X	X	X	X	X	X	X
"What is the current activity?"			+								

	DRATS02	MDRS03	MDRS04	MDRS05	DRATS05	CDS05	PA06	iMAS06	DRATS06	POGO07	iMAS08
Start <selected activity from plan>			+	X	X	X	X	X	X	X	X
"Start first activity"				+							
Plan distribution to mobile agents				+	X	X		X	X	X	
Query activity and remaining time				+	X	X	X	X	X	X	X
Allow creating new activity of a type (e.g., sample, survey, walk) at <location>				+	X	X	X	X	X	X	X
"What time is it?"				+	X	X	X	X	X	X	X
Display <procedure>						+	X				
"Read the <Nth step> of <procedure>"						+	X				
"What is <astronaut>'s current activity?"							+		X		
Display {PowerPoint | JPG}									+		
Display next/previous page									+		
Zoom in/out									+		
Pan display left/right									+		
Alert about walk back emergency										+	
Science Data Logging											
Create and number sample bags	+	X	X	X	X	X		X	X	X	X
Record a voice note	+	X	X	X	X	X		X	X	X	X
Play a voice note assoc. w/ location or sample		+	X	X	X	X		X	X	X	X
Associate sample bag or voice note with location		+	X	X	X	X		X	X	X	X
Photographs logged by time & location		+	X	X	X	X		X	X	X	X
Science data transmitted to Hab and stored on habitat agent platform (HabCom)		+	X	X	X	X			X		
Science data emailed to RST (enable/disable)		+	X	X	X	X	X		X	X	X
Download an/all image(s) from camera		+	X	X	X	X		X	X	X	X
Science data stored in hierarchical database organized by EVA, astronaut, location, time, & data type on web pages accessible remotely during EVA (includes voice notes & images)			+	X	X	X			X	X	
Define pair-wise association of voice notes, sample bags, & images			+	X	X	X		X	X	X	X
Sample bags, voice notes, and photographs auto-associated with activity of EVA plan			+	X	X	X		X	X	X	X
Name the last image or collection			+	X	X	X		X	X	X	X
Replay voice note <number>			+	X	X	X		X	X	X	X
Name sample bags by pattern LL/DD/DD (allowing use of ICA alphabet)			+	X	X	X		X	X	X	X
Allow references to "last image" and "last voice note"			+	X	X	X		X	X	X	X
Automated printing of sample bag label				+							
Reference image collections by number or location/bag association				+	X	X		X	X	X	X

	DRATS02	MDRS03	MDRS04	MDRS05	DRATS05	CDS05	PA06	iMAS06	DRATS06	POGO07	iMAS08
Allow references to "image of <bag>|<location>"				+	X	X		X	X	X	X
Reference panoramic images by number, location, or "last"				+	X				X		
List locations, sample bags, images, voice notes, collections, panoramic images {near <location>} on demand				+	X	X		X	X	X	X
Science data accessed from dynamic EVA map (TerraServer)					+	X			X		
Enable/Disable automatic data associations											+
Voice Mail											
"Tell <astronaut> … <any speech> {at <time>}							+				
"Play voice message from <astronaut>"							+				
"Continue last voice note"							+	X	X	X	X
"Send <voice note> to <person>"									+		
Alert Management											
Enable/Disable all alerts		+	X	X	X	X	X	X	X	X	X
Enable/Disable last alert		+	X	X	X	X	X	X	X	X	X
Enable/Disable selected alert for <astronaut>			+	X	X	X	X	X	X	X	X
Enable/Disable all alerts of type X			+	X	X	X	X	X	X	X	X
Enable/Disable explicit confirmation of command interpretation				+	X	X	X	X	X	X	X
Enable/Disable conditional alerts (e.g., "Tell me/<person>/everyone when the generator comes online.")							+				
Acknowledge command acceptance via beep							+	X	X	X	X
Voice Command Controls											
Increase/Decrease Volume				+	X	X	X	X	X	X	X
"Say that again"/"Repeat that"				+	X	X	X	X	X	X	X
"Stop talking"				+	X	X	X	X	X	X	X
"Are you listening?"				+	X	X	X	X	X	X	X
"Start/Stop Listening"							+	X	X	X	X

Appendix IV. Organization of Data Analysis Table

The following table provides an overview of the data available and what was calculated for evaluating the agent-based open architecture.

(Key: **KSLOC** = Thousand Source Lines of Code; **Workflow Agents** are written in the Brahms programming language, sending messages to each other and to **Communication Agents** (CA), which are written in Java and usually call external APIs written in Java, C++, etc. "Packaged" refers to the entire system configuration, which includes multiple instances of an agent type.)

PERSONNEL	**CONTRIBUTION**
ARC Co-Investigators	Project Lead, Scenarios, Field Coordinator
	Project Manager, Scenarios, Integration
Brahms Modelers	Workflow & CA Agents
ARC Programmer	ScienceOrganizer
ARC MEX Team	MEX & GPS, Nonin sensor
ARC Programmer	Voice Commanding (RIALIST)
Audio Manager	CAIPack HUD, & Suit Audio
Student Intern	Stanford Biovest
JSC ERA Team	ERA(s) & SCOUT Integration
ARC IRG Team	K9 Rover, Gromit, EUROPA
JSC/HRP Scientist	MRA EXCEL & Suit integration
Project Time Data	**PROGRAMMING FTE DURING PERIOD**
	Project Development Start Date
	Project Completion Date
	Elapsed DDT&E Days
Effort Data	Annualized PROJECT FTE
	FISCAL YEAR
	Dialog Agent Programmer FTE
	Workflow Team FTE
	FY FTE (Workflow Agent Team)
	# Programmers
	FY FTE (all projects)
Software Data	**Workflow Agent Types Count**
	Number of NEW WF Agent Types
	Number of Modified WF Agent Types
	Number NEW + Modified WF Agent Types

New WF Agents KSLOC
Unchanged WF Agents KSLOC
Modified WF Agents KSLOC
Eliminated WF Agents KSLOC
Modified WF Agents Carried Over KSLOC
Modified WF Agents KSLOC ADDED
(Verify KSLOC Calculations = Total)
All Workflow Agents KSLOC
% New WF Agents KSLOC
% Unchanged WF Agents KSLOC
% Modified WF Agents KSLOC
(Verify % Calculations = 100%)
% WF Agent KSLOC New or Modified
Workflow Agents Packaged Count
Workflow Agents KSLOC Packaged
Communication (Integration) Agent Types Count
Number of NEW Comm Agent Types
Number of Modified Comm Agent Types
Number NEW + Modified Comm Agent Types
New Comm Agents KSLOC
Unchanged Comm Agents KSLOC
Modified Comm Agents KSLOC TOTAL
Eliminated Comm Agents KSLOC
Modified CA Carried Over KSLOC
Modified CA KSLOC ADDED
Dialog CA KSLOC
Added KSLOC to Previous Dialog CA
Added KSLOC to Carried over (modified) Comm Agents excluding Dialog CA additions
(Verify KSLOC Calculations = Total)
All Comm Agents KSLOC
% New CA KSLOC
% Unchanged CA KSLOC
% Modified CA KSLOC
(Verify % Calculations = 100%)
% CA KSLOC New or Modified
% Modified CA KSLOC attributed to Dialog Agent Modifications
Communication Agents Packaged Count
CA KSLOC Packaged
Number of All Agent Types (Workflow + CA)
Change in number of all Agent types
TOTAL All Agent Types KSLOC
TOTAL Configuration Package KSLOC
Change in all Agent Types KSLOC
CHANGE IN TOTAL PACKAGED KSLOC

Productivity Data	PRODUCT LINE INHERITANCE
	Number New Capabilities
	Number of Peripheral Systems Added
	Number of Requests/Alerts Added
	Number of Request/Alert Capabilities
	Capabilities added/FY FTE
	New + Added Workflow Agent KSLOC/Workflow Team FTE
	New + Added CA KSLOC/All Teams FTE
	New + Added All Agents KSLOC
	New + Added KSLOC Grouped by Projects
	New + Added KSLOC/All Teams FTE, excluding Dialog CA Programmer
	New + Added All Agents KSLOC/All Teams FTE
	New + Added All Agents for Grouped Projects/All Teams FTE
	New + Added All Agents Grouped Projects/# Programmers
	Change in CA Packaged SLOC/FY FTE
	Change in all Agent Types SLOC/FY FTE
	Change in Package KSLOC /FY FTE
Cost Data	**Estimated Software Budget ($000)**
	Cost ($000)/Total Agent Types KSLOC
	Cost ($000)/Total Packaged KSLOC

Appendix V. Completing the Picture: System Specification through a Goal-Oriented Control Framework

The lunar surface system capabilities envisioned will almost certainly be realized as a system-of-systems developed by a multinational, multi-agency set of partners. The open architecture described thus far addresses many of the most salient features of such systems, as well as the challenges associated with the planning, developing and integrating of an incrementally-delivered, distributed system. The open architecture provides an infrastructure for "plug-and-play" interoperability, scalability and maintainability.

While an open architecture provides an important set of features, delivering and operating this lunar surface system-of-systems has a residual number of challenges that can be met through adoption of a complementary set of architectural principles and approaches; the principles of Goal-Oriented Control coupled with model-based systems engineering practices.

A real-time, distributed, safety-critical system with heterogeneous components (which in many cases have been developed by multiple teams) is a sure recipe for complexity. This situation warrants an architectural framework which explicitly maps well-stated properties of interest to the design elements responsible for satisfying them. This framework needs to be rich enough to cover the concepts of the domain of interest while ideally facilitating systems engineering best practices (in particular specification and knowledge capture, analysis, verification and validation). Goal-oriented Control is such a framework.

Goal-Oriented Control

Goal-oriented control is an architectural framework for specifying a system based on explicit statement of desired system behavior (which is defined as constraints on states of the system of interest) and progressively defining the elements to best realize the aforementioned specifications. Along with a conceptual model, the framework affords a set of techniques and software tools.

The goal-oriented control approach parallels the systems engineering best practice of grounding design in well-stated, verifiable and traceable requirements. *Goals* are defined as requirements that specify a constraint on a *state variable* of interest. These state variables are values that can change over time, and must be controlled in order to meet system objectives. Goals are progressively elaborated into supporting goals until they are atomic statements that can be controlled or measured by a single element and readily verified.

As part of specifying our goals, we also specify the attendant control system to achieve the goals. This approach is a close analogue to the practice of classic control theory where a plant description (what you want to control) is coupled with a set of sensors, actuators and estimators to realize a desired set of global system properties. Designers start by specifying what properties the system as a whole must exhibit, then they look at

the system which must be controlled along with the appropriate influences of the environment affecting the plant while determining the necessary control system (sensors, actuators, estimators) to deliver the desired system performance.

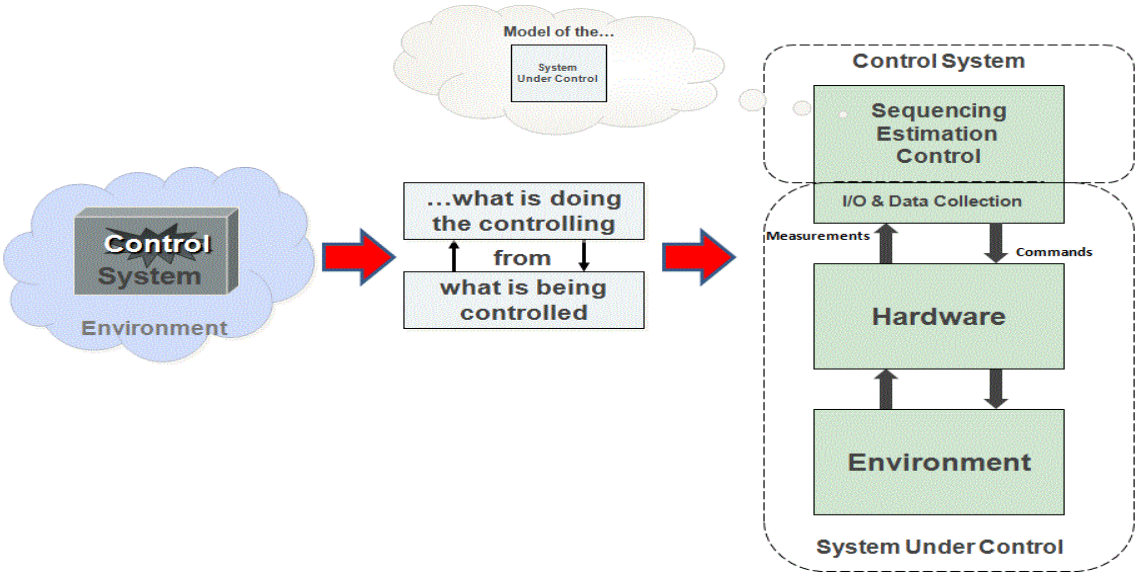

Figure 22. Identifying the system under control and how it fits in the operating environment.

The figure above illustrates the basic analytic process of identifying *what is the system under control* and *how does it fit in the operating environment*. Once identified, the practitioner has a firm basis to begin the requirements process; defining goals as states of interest with necessary constraints.

The system under control is ultimately the system hardware operating in the operational environment. The control system (schedulers, estimators, controllers, sensors) is comprised of networked distributed software agents. The open architecture described previously provides the infrastructure for these distributed agents to work together to achieve the specified system goals.

Models, along with measurements and command histories, provide the control system with the state knowledge it needs to act appropriately in all operational contexts. The models used to specify the system in requirements elicitation and system development are carried over into operations for use in running the actual system, reducing the gaps between concept, design and operations.

The Benefits of Goal-Oriented Control

The benefits of the Goal-Oriented framework fall under two related though distinct categories, *technical* and *process*. The *technical* benefits describe the intrinsic quality premiums of the realized systems, while the *process* benefits cover the advantages of the method for addressing project cost, technical and schedule risks.

Technical Benefits
Goal-oriented control systems allow the user to specify system behavior as explicit and verifiable constraints of defined states. As a result, systems have a run-time guarantee of "correct" (in accordance with specification) behavior. In contrast with a more traditional procedural implementation, the goal-oriented system "keeps track" of system states and acts in a closed-loop manner to issue the appropriate commands in to satisfy the operator-provided goals. As a related benefit, the system also uses its models and measurements to project states into the future to determine if planned goals can be met based on what has happened so far. This facilitates a much more nuanced, robust way of specifying system behavior. This also results in a more integrated and transparent fault protection; fault protection is rightfully integrated as part of overall behavior management.

The goal-oriented control approach also scales nicely to arbitrarily complex numbers of distributed elements. As long as the user can state goals as constraints on any time-varying property, a system and its behavior can be specified by using the small set of concepts, relationship and patterns of the goal-oriented framework.
The simplicity of the framework also allows flexibility of implementation with regards to the level of autonomy. While the control system has a running model of the system under control which it updates through measurements, it is up to the designer to determine what commands (if any) the system can issue. For instance, the control system can use its model information in an advisory role to a human actor, suggesting commands while the human provides the ultimate authority to proceed.

Process Benefits
As mentioned previously, the lunar surface system-of-systems will feature a set of elements developed by a number of different teams and delivered in an incremental fashion. Well-formed, unambiguous, system specification through models is becoming an indispensible tool in the systems engineering and architecture of such systems. Models help by providing developers, testers and users a common understanding of what the system should do, how and why. By using a common representation, the program fielding the system has a *lingua franca* to discuss and work system issues. Goal-oriented control, with its innate emphasis on modeling, capitalizes on these benefits. Not only do the models inform the design process, aiding in the discovery and analysis of requirements, the models also serve as the "smarts" for the fielded system, providing the capability for true closed-loop, cognizant control.

Documentation of the system knowledge as models provides a "living record" of issues, concerns and decisions and ties these artifacts into the actual flight code. The benefits to maintenance and upgrades over the life cycle of such a comprehensive record are manifest. Developers and stakeholders have a more unambiguous means for specifying the system, testers have better requirements to test and operators have a more natural, powerful language for specification of behaviors for robust execution.

Goal-oriented models also enable developers and testers to employ powerful analytic means (both simulation and static "theorem provers") for analyzing and verifying designs. Model-checking applications are seeing significant use in checking mission-

critical systems in many domains, both improving system safety and reducing development costs. The well-specified goals of the goal-oriented approach and the well-defined elements and relations of the framework are key to implementing these techniques. Formal system verification though models allows testers to check a range of design properties for multiple operational contexts economically.

Appendix VI. Glossary

activity	An extended behavior of some agent occurring in some time-space context, e.g., "living on the lunar surface." Oriented by motives, which often include goals with tasks to accomplish. See Clancey (2002).
agent	Located, proactive software program with models of work and local situation; communicates messages with other agents
autonomous	A mode of automated operation in which a system is programmatically controlled to pursue (and possibly reformulate) goals without additional human guidance.
API	Application Programming Interface
Brahms Work Practice Modeling and Simulation System	Multi-agent discrete event simulation system designed for simulating human work practices (situated activities), represented as chronological, located behaviors with interactions among people and automated systems and tools.
Collaborative Infrastructure	Environment for hosting distributed software applications and components providing services that can be used to simplify management, status monitoring, network transport, messaging, data distribution and translation; CI generalizes the CORBA/Speech Act method of information exchange developed for the Mobile Agents exploration systems. Currently used in OCAMS for ISS file management (Figure 21).
Communication Agent	JAVA program that translates between the programming language objects and methods of a component API and Speech Acts, represented in the language of the task domain.
component	Hardware or software system interoperating with other components in the exploration system.
domain model	Structured representation for some purpose of some system and/or processes, e.g., a geographic model of a lunar region; a

	model of an EVA traverse; a model of an exploration system configuration
EVA Robotic Assistant (ERA)	Rover designed for science support (Hirsh, et al. 2006)
exploration system configuration	Particular set of integrated hardware and software that provides support to crew for some purpose (e.g., science, survival, mobility, general inquiry of some region)
FIPA	The Foundation for Intelligent Physical Agents (FIPA) developed computer software standards for heterogeneous and interacting agents in agent-based systems. Replaced by IEEE Standards Committee in 2005.
information	Interpreted data; more specifically: Perception of some condition or conceptual/inferential interpretation of the meaning of perceived data; e.g., the meaning of a spoken utterance.
Information Exchange Service	Functional capability in a software architecture that constructs and/or conveys information by gathering, transforming, interpreting, packaging, etc. data, information, and/or commands
RIALIST	Software program capability of recognizing and generating spoken utterances; uses a grammar of valid utterances and their Speech Act interpretation; related to NUANCE commercial products.
Speech Act	Broadly, the implicit meaning of an utterance, viewed as a performance by the speaker, e.g., promise, command, warning. Voice commands are implemented by workflow agents to satisfy the intention agreed by the crew and exploration system developers, e.g., “Boudreau, Stop!” means “stop immediately, don’t ask for confirmation.” Thus, the “stop” utterance is a different kind of speech act than the “go to waypoint N” utterance, which requires confirmation.
task	A functional unit that describes goal-oriented action; contrasted with “activity.”

workflow system	In the context of the space program, an exploration system designed to automate the creation, storage, and communication of information among system components, which may be operating “autonomously”; usually by interacting with astronauts and/or remote support teams.