NASA/TP—2012–216040

Lunar Surface Systems Software Architecture Study: Interoperability

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

Michael Lowry
Ames Research Center, California

August 2012

NASA/TP—2012–216040

Lunar Surface Systems Software Architecture Study:

Interoperability

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

Michael Lowry
Ames Research Center, California

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. Contributions and review were provided by Robert Nado and Maarten Sierhuis (ARC) and Edwin Crues, Dan Dvorak, and Grailing Jones (JPL). 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.

TABLE OF CONTENTS

FIGURES

TABLES

1 EXECUTIVE SUMMARY

Interoperability refers to "the ability for two or more systems to exchange information and to use the information that has been exchanged." This report focuses on interoperability for surface exploration involving a diversity of hardware and software systems that are dynamically interacting in a mobile, distributed environment.

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). It elaborates a companion report, *LSS Software Open Architecture Study* (Clancey et al. 2010b), which systematically analyzed data from the Mobile Agents Project, part of NASA's Intelligent Systems and Human-Systems Integration Exploration Technology Development Programs (2002-2006); the first report quantified the effort involved in extending and reconfiguring components within an open architecture framework.

In practical terms, the purpose of this report is to begin to answer the question: What do different vendors (including international partners) need to put into their systems to realize interoperability for exploration systems? Or in more technical terms: What are the functional requirements, independent of specific scenarios or workflow capabilities, that will enable interoperability in commanding, data access, and security—for any purpose, including science, medical, resources, operations, etc.—across a wide range of component technologies, exploration system configurations, and settings?

This report explains what is meant by "workflow" and how it relates to interoperability. Sophisticated workflow functions include the heuristic guidance of people, robotic systems, and software agents to facilitate cooperative work (e.g., parceling tasks among people and robotic systems and delegating information processing to software agents). Such systems use available information to plan, alert, guide, record, and control the ongoing flow of information and commanding to assist or directly orchestrate inter-operation.

Model-based inference (aka "reasoning") is the fundamental computational method enabling workflow automation. Specifically, models of exploration systems are used by programs, often called "intelligent systems," to monitor, predict, interpret/explain, control, diagnose, and plan inter-operation of subsystems. By implementing these capabilities in a flexible way, exploration systems can increase efficiency through automation, reliability through being adaptive, safety through monitoring and alerting, and extensibility through co-operation among new and existing LSS elements.

Interoperability also enables distributed access to sensor and vehicle state interpretations, as well as closed-loop command and control automation, with or without people in the loop. Overall, the exploration system is more flexible by enabling a wide range of "point of control" (PoC) configurations for LSS elements, facilitating crew situation awareness and control. Furthermore, PoC migration and/or distribution enables dynamic reconfiguration of system roles and oversight, including variable autonomy (Alena 2010).

In general, workflow automation involves associating, packaging, and communicating both discrete and continuous data streams; the purpose may be archival or for ongoing operations, by either people or automated systems. For example, EVA photographs are more efficiently organized for later interpretation if the database automatically pulls together data about location, time, camera source, and the context of the EVA activity. This is accomplished by synchronizing and often reformulating data from diverse sources at runtime, creating new information representations (e.g., web pages, text summaries).

This report begins with a short review of different perspectives on interoperability and its relevance to exploration systems, expressed as figures of merit for evaluating interoperability, largely derived from recent critical analyses of the ISS computing architecture: safety, security, data management, incremental buildup, productivity, and cost (Section 3). A detailed emergency EVA scenario illustrates advantages and methods for state-of-the-art interoperability by automating workflow (Section 4). We conclude from analyzing the EVA emergency scenario that a Lunar Surface System with workflow automation would significantly improve safety and productivity by enabling a diversity of distributed subsystems to interact flexibly (e.g., habitat, pressurized vehicle, spacesuit, robot, instruments) (Section 4.5). These workflow functions include real-time, closed-loop coordination of multiple subsystems; consolidating, filtering, and abstracting data; creating and forwarding relevant information; and assembling data from multiple subsystems to construct commands for devices/applications.

We then distinguish low and higher-level interoperability in the EVA scenario, clarifying the limitations of a systems integration perspective that focuses on connectivity and communications. The physical "building blocks" metaphor is contrasted with a process metaphor of a distributed network architecture with control layers (Section 5). Candidate software architectures for interoperability are compared relative to the figures of merit (Table 2), with the distributed network control architecture found to be superior by all metrics. The recommendations are then related to the present C3I Architecture (Figure 6.1) and enduring challenges are discussed. An appendix (Section 8) presents an example of an agent-based implementation of workflow orchestration.

The trade study also highlights that multiple architectural perspectives are useful for designing a complex integrated, distributed system, including networking, inter-process communication, interfaces, and workflow. In practice, a distributed network architecture will in some respects involve building blocks and functional layering. Analysis shows that the distributed network perspective is more embracing by enabling multiple design perspectives to be related into a reconfigurable and robust operational system of systems.

The comparative analysis of interoperability architectures presented here complements the *Real-Time Avionics Middleware Architecture Trade Study* (Alena 2010) by clarifying how certain interoperability functional requirements, including dynamic reconfiguration and workflow automation, go beyond common middleware capabilities. In practical implementations, middleware provides a medium for communicating data and commands among devices and applications, while workflow agents orchestrate the flows. The kind of interoperability required within a system of systems depends on whether and how particular components interact:

- *Low-Level Interoperability (Middleware)*—for systems whose components operate independently and/or interact only in fixed ways (e.g., functional hierarchy) and tend to be of fixed types (e.g., plug-ins).
- *Higher-Level Interoperability (Workflow Automation)*—for systems whose components are distributed, heterogeneous (i.e., not designed to a single standard), and are intended to interact with other software and/or hardware (e.g., a planner subsystem; a database; an Email program).

In summary, interoperability shifts the systems engineering notion of "integration" from connectivity and forwarding telemetry and commands between subsystems to *orchestrating operations in a system of systems to create and manage information.* Interoperability is concerned with enabling higher-order processes to control and hence coordinate actions of subsystems within the ongoing context of human tasks (work).

The legacy systems of the International Space Station provide a useful example for considering to what extent a "clean slate" is desirable, rather than designing around legacy systems. Why not simply replace them by redesigned hardware and a modern software framework? Simply put, interoperability involves managing a diversity of technologies in a complex, already functioning system. In part, diversity stems from technological advances that make today's state-of-the-art systems "legacies" in a few years. Although we might say that incremental buildup is desirable because we cannot afford to build everything at once, incremental buildup is also inevitable because we cannot afford to replace everything at once when technology advances. We will always need to deal with legacy systems that generate, represent, and store data in different ways, as well as upgrades that seek to use existing data and control legacy systems in new ways. Inevitably, operating systems and computational platforms will continue to evolve. Rather than a "clean slate" approach, we recommend a framework that is designed for reusability, extensibility, and scalability.

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 overarching 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.

This report focuses on interoperability for surface exploration involving a diversity of hardware and software systems that are dynamically interacting in a mobile, distributed environment. This report elaborates a companion report, *LSS Software Open Architecture Study* (Clancey et al. 2010b), which systematically analyzed data from the Mobile Agents Project, part of NASA's Intelligent Systems and Human-Systems Integration Exploration Technology Development Programs (2002-2006); the first report quantified the effort involved in extending and reconfiguring components within an open architecture framework.

This report presents a surface EVA scenario based on state-of-the-art capabilities to show how workflow automation drives functional requirements for interoperability. A basic trade analysis of alternative well-known computing architectures shows the advantages of distributed network control; Section 8 presents an example agent-based implementation. The comparative analysis complements the *Real-Time Avionics Middleware Architecture Trade Study* (Alena 2010) by clarifying how certain interoperability functional requirements, including dynamic reconfiguration and workflow automation, go beyond common middleware capabilities.

In practical terms, the purpose of this report is to begin to answer the question: What do different vendors (including international partners) need to put into their systems to realize interoperability for exploration systems? Or in more technical terms: What are the functional requirements, independent of specific scenarios or workflow capabilities, that will enable interoperability in commanding, data access, and security across a wide range of component technologies, system configurations, and settings?

It is often also asked how the upfront cost of adopting an interoperability framework can be justified. This report illustrates how state-of-the-art systems engineering architectures, relying on advanced computational methods, enable a form of interoperability that provides increased safety, extensibility, reusability, and productivity—yet with reduced costs and effort required for constructing these more capable systems. That is, benefits accrue and are also realized in every system constructed.

Interoperability can be viewed as occurring within a system conceived as a single operating entity, such as a robotic system or life support system constructed from multiple, interacting components. The analysis of architectures in this report is relevant to a single entity, but our emphasis is on communications and coordination among multiple, semi-autonomous "entities," such as a habitat, pressurized rover, spacecraft, spacesuit, etc. In this respect, with people distributed on Earth, on a planetary surface, and in space, characterizing inter-operation necessarily focuses on what people are trying to accomplish (their "work") and how they do this by interacting with hardware and software systems. In general, the work of mission operations constitutes a choreographed sequence of operations of individual people, organizations, and mechanisms; the exchange of data, information, and records in any form (documents, photographs, charts) constitutes the flow of their work.

This report explains what is meant by "workflow" and how it relates to interoperability. Some workflow capabilities relevant to surface EVAs include:

- automating data correlation, interpretation, and structured presentation;
- creating plans and monitoring plan execution;
- configuring routes and dynamic navigation guidance;
- diagnosing malfunctions and presenting repair procedures;
- transforming and executing higher-order commands by controlling subsystems;
- relaying data, alerts, commands, and plans, etc. among people, robots, and software subsystems (e.g., databases).

Model-based inference (aka "reasoning") is the fundamental computational method enabling these capabilities. Specifically, models of exploration systems are used by programs, often called "intelligent systems," to monitor, predict, interpret/explain, control, diagnose, and plan inter-operation of subsystems. By implementing these capabilities in a flexible way, exploration systems can increase efficiency through automation, reliability through being adaptive, safety through monitoring and alerting, and extensibility through co-operation among new and existing LSS elements.

Interoperability also enables distributed access to sensor and vehicle state interpretations, as well as closed-loop command and control automation, with or without people in the loop. Overall, the exploration system is more flexible by enabling a wide range of "point of control" (PoC) configurations for LSS elements, facilitating crew situation awareness and control. Furthermore, PoC migration and/or distribution enables dynamic reconfiguration of system roles and oversight, including variable autonomy (Alena 2010).

This report begins with a short review of different perspectives on interoperability and its relevance to exploration systems, followed by a concrete scenario to illustrate advantages and methods for state-of-the-art interoperability automating workflow. We then compare low and higher-level interoperability, clarifying the limitations of systems integration focusing on connectivity and communications. The physical "building blocks" metaphor is contrasted with a process metaphor in a distributed network architecture with control layers. Candidate architectures are compared relative to figures of merit largely derived from recent critical analyses of the ISS computing architecture. Interoperability recommendations are related to the present C3I Architecture and enduring challenges are discussed. An appendix provides an overview of how workflow processes are orchestrated in a multiagent software architecture.

3 Relevance of Interoperability to LSS

3.1 Types of Interoperability

Interoperability refers to "the ability for two or more systems to exchange information and to use the information that has been exchanged" (CxP 70022-01, p. 7; Wikipedia cites the IEEE Glossary). A more general definition of interoperability includes organizations working together:

> Interoperability is a property referring to the ability of diverse systems and organizations to work together (inter-operate). The term is often used in a technical systems engineering sense, or alternatively in a broad sense, taking into account social, political, and organizational factors that impact system to system performance.[1]

In effect, the technical and broader perspectives correspond to low-level (syntactic) and high-level (semantic) interoperability.

> If two or more systems are capable of communicating and exchanging data, they are exhibiting *syntactic interoperability*. Specified data formats, communication protocols and the like are fundamental. In general, XML or SQL standards provide syntactic interoperability…. Syntactical interoperability is a necessary condition for further interoperability.
>
> Beyond the ability of two or more computer systems to exchange information, *semantic interoperability* is the ability to automatically interpret the information exchanged meaningfully and accurately in order to produce useful results as defined by the end users of both systems.

In the context of LSS, the systems that might interoperate (also called elements or components) are most obviously vehicles, space suits, habitat/modules, robots, and associated power systems. These systems usually include and/or are controlled by software applications and interfaces (including data systems and mission operations systems) by which people and system components communicate goals and information. We call this combination a *surface exploration system*.

It is sometimes supposed that semantic interoperability requires "a common information exchange reference model," in which "the content of the information exchange requests are unambiguously defined: what is sent is the same as what is understood."[1] In particular, Constellation Program documentation has suggested that a "common information structure and language [is] necessary for systems to perform appropriate command and control functions using the information exchanged" (CxP 70022-01, p. 7).

However, information exchange and mutual interpretation is only a functional requirement of the overall system's operation; it need not imply that every system component must "speak the same language." Instead, as explained in the companion report, *LSS Software Open Architecture Study* (Clancey et al. 2010b), semantic interoperability can be achieved by using a method of dual application programming

[1] Interoperability. (2010, September 18). In Wikipedia, The Free Encyclopedia. Retrieved 16:34, October 8, 2010, from http://en.wikipedia.org/w/index.php?title=Interoperability&oldid=385488394

interfaces (APIs).[2] That is, interoperability can be achieved through the interaction of workflow processes that orchestrate application/subsystem inter-operation. These workflow processes speak a common information structure and language, providing a common mechanism for communicating between systems. This workflow architecture makes it unnecessary for every component to literally use a single information structure and language to perform command and control functions. Section 8 provides details about how an agent-based architecture can be used to implement this dual API method of information exchange.

Our concern in this report is with an open software architecture that promotes interoperability for semantic interoperability, that is, with respect to goals, information, and plans, including command sequences. Such communication is more abstract than communication in terms of telemetry, status, measurements, and low-level commands, though it may involve forwarding them. Semantic interoperability can be viewed as a higher-level form of open architecture, enabling a co-operating "system of systems."

3.2 Figures of Merit for Evaluating Interoperability

An architecture enabling interoperability is motivated by design criteria ("figures of merit") for exploration systems. More specifically, certain LSS design criteria (figures of merit) are satisfied by workflow automation; automation in turn is enabled by certain functional requirements on interoperability that can be provided by a *service-oriented architecture* (Section 5.2):

Design Criteria for LSS Exploration Systems (e.g., crew self sufficiency)

↑

Workflow Automation Capabilities

↑

Interoperability Functional Requirements

↑

Service-Oriented Architecture

In particular, workflow automation can be used to satisfy design requirements among LSS subsystems (e.g., databases, software applications) for security/privacy; safety/accuracy and timeliness; data management; incremental buildup/extensibility and scalability; operational efficiency/productivity; and cost of Design, Development, Testing, and Engineering.

Examples of such LSS design requirements and their relation to system architecture include:

[2] In this scheme, a secondary, "communication" API translates from operations expressed in a higher-level, task-oriented language to the language of a component API (which exposes the internal objects and processes of a device/subsystem).

- **Security/Privacy:**
  - Requirements for security derive in particular from an international partnership, which requires some shared access to subsystems that must be controlled (e.g., telemetry and commanding devices over networks). Providing a capability for opportunistic reconfiguration including remote commanding requires flexibility for reprogramming access and controls.
  - Security concerns also derive from public access to mission progress and (limited) science data.
  - Medical data and crew personal records (e.g., email, photographs) and communications are necessarily regulated, in some cases by privacy laws, requiring even within NASA's networks controlled access to computers and archived data.

- **Safety/Accuracy/Timeliness:**
  - Exploration is inherently difficult because we cannot precisely define mission needs a priori. Flexibility and adaptability will often be required.
  - Space medicine in particular has interoperability requirements for electronic exchange of information among crew and ground support via a potential diversity of devices, networks, and programs to improve the safety of surface operations (HIMSS-EHRVA 2005).
  - During a mission a great deal of human expertise is remote, with access to more sophisticated instruments than available to the crew. In particular, flight surgeons must use telemedicine technology to observe, diagnose, and treat the crew.
  - An analogous situation occurs with scientists on Earth who could program a robotic laboratory that provide reconnaissance for EVAs or pursues different or more detailed data after an EVA. Scientists provided with sufficient information may be able to guide later activities during an EVA, despite time delays.

- **Data Management**
  - Different uses of data (medical, scientific, engineering) may require transforming data, information, and models in different ways, and communicating with different detail and urgency.
  - Data from different devices must be integrated for interpretation (alerting, diagnosis, maintenance), presentation, and storage. Multiple standards and manual processing result in lack of time synchronization of recordings from multiple devices and inaccuracies.
  - Data may be unnecessarily detailed or excessive for transmission to Earth. This problem already occurs for ISS for medical data and for planetary robotics. Rather than bulk transmission over limited and expensive bandwidth, automated culling of what's worth analyzing can use human expertise and networks more efficiently (e.g., MER software analyzed photographs for dust devils and clouds, transmitting only likely candidates). An analogous situation occurs for engineers in monitoring and diagnosing remote systems (e.g., life support and tools/machinery).

- **Incremental Buildup/Scalability/Extensibility:**
    - Requirements for extensibility derive especially from both national and international development of assets and improved systems over time, including increased complexity of automation.
    - Interoperability enables a new system to be added to an existing collection of space assets so that they can effectively operate together. For example, a European habitat could be added to an existing US lunar base and be able to interface and operate effectively.
    - In some respects, the focus is on "local evolvability," in which integrated components have multiple uses and are reconfigured into a customized "system of systems" for different missions and activities (e.g., EVAs) during a mission.

- **Operational Efficiency/Productivity:**
    - Interoperability enables workflow automation, which increases crew productivity.
    - Co-operation of systems can also enable gathering information efficiently in what planetary missions call "coordinated science." For example, an EVA may involve sampling and/or drilling to refine a planetary geology model. By combining a geochemical analysis of the samples with orbital surveys enables relating the local data to regional patterns, perhaps enabling an explanation of the broader geomorphology and its evolution, reducing the number and complexity of future traverses in that region.

- **DDT&E Cost:**
    - Designing for inter-operability at first requires more complicated architectures and may require more effort, but it provides all of the above advantages for meeting mission needs.
    - Software development cost may also be reduced by using an architecture promoting inter-operability by virtue of:
        - Easy and well-defined connectivity and communication among subsystems, including built-in support for a variety of protocols
        - Efficient reuse of a library of services include data exchange, information transformation, and methods for model-based interpretation, monitoring, prediction, and control
        - Ease of developing higher-level workflow automation that builds on existing lower-level data processing and subtask automation

State-of-the-art software architectures enabling interoperability have been developed in research and commercial settings, including particularly in NASA's ETDP research. The companion report, *LSS Software Open Architecture Study* (Clancey et al. 2010b), analyzes data demonstrating advantages for safety, extensibility, productivity, and cost.

The following sections of this report explain in more detail how a workflow automation approach can address the figures of merit for evaluating exploration systems, the requirements workflow places on interoperability functions, and how these requirements can be implemented in different architectures. Our approach is to begin with a concrete scenario, which illustrates several of the operational advantages.

4 Scenario: Emergency Return to Habitat During Surface EVA

4.1 Scenario Objective

This surface EVA medical scenario combines an analysis of operations concepts and data management for surface "medical conditions,"[3] analysis of difficulties with current medical computing onboard ISS[4], and interoperability capabilities of state-of-the-art exploration technology (Clancey, et al. 2005, 2006, 2007; Dowding et al. 2006; Hirsh et al. 2006; Johnson et al. 2009; Johnson 2010). The scenario emphasizes how automated workflow capabilities enhance crew self sufficiency in the context of a medical emergency. Based on the automation specified in the scenario, we derive interoperability functional requirements, and then work out trades on alternative architectures/implementations.

4.2 Scenario Assumptions

The scenario involves a medical problem compounded by engineering failure during a science EVA. We assume the following:

- At least two crewmembers are on a surface pedestrian EVA; four crewmembers are in the habitat-spacecraft (Hab), including the physician.
- The planetary body is a small moon or NEO, at least 100 km in diameter.
- The crew is towing along a sample and equipment carrier, analogous to that used on Apollo 12.
- A medium-large robotic science laboratory (MSL-class) is within 50 meters; it is intended for permanent surface deployment.
- The terrain is hilly and complex, with some hazards and obstacles (e.g., ravines, craters).
- The EVA is scheduled for four hours and has two hours remaining.
- The Hab is station-keeping near the surface.
- The two EVA astronauts have been laboring to reach outcrops and/or foray a bit into craters to take measurements and samples.

[3] NASA, Space Medicine Division and Space Life Sciences Directorate, Johnson Space Center. 2009. "Exploration medical conditions concepts of operations." Draft: Dated 9 October 2009.

[4] NASA, Medical Informatics and Health Care Systems Division SD4, Johnson Space Center. 2010. "International Space Station: Next generation Crew Healthcare System (CHeCS)—Common crew health data management and communications system concept." Draft presentation: Space Medicine CHeCS MEC-to-Common Data Management & Communications Transition Plan Options.ppt.

4.3 Scenario Narrative

About halfway through the EVA a medical alert is broadcast by verbal warning and visual alarm on the heads-up display (HUD) of one of the astronauts (EVA1), indicating that his metabolic rate exceeds nominal.

Simultaneously, the IVA physician receives the same alarm through the loudspeaker in the habitat. She and the other three crewmembers confer briefly. They have experienced these alarms on about a third of their EVAs, so are familiar with the medical monitoring intent. They review the EVA Log web page, which is being updated automatically during the EVA to show plan, schedule, progress, and data collected. They also study the live video feed from EVA1 and EVA2's head cams. They decide that the overall plan and situation appear nominal, and EVA1 can be trusted to adapt. One of the crewmembers notices that there will be a more demanding hill to climb in the last third of the EVA; the physician suggests to EVA1 to take it easy. (The alarm has also been transmitted to Earth by Email to flight surgeon PDAs. It will not arrive for 35 minutes, meaning a response would only arrive at best when the crew would be already returning to the Hab.)

Given the timing and location and goals, the EVA crew proceeds, though the astronaut who received the warning attempts to slow down, allowing the other astronaut to do more of the bending and carrying.

However within 15 minutes a second alarm is sounded, this one indicating that the metabolic rate is still excessive and now the life support consumables are within 90% of the minimum that will prevent completing the scheduled EVA. EVA1 notices now that he is indeed tired and suggests a short rest in place, intending to take a drink and catch his breath.

At this point a second anomaly develops. A combination of problems is known to be a likely cause for serious emergencies. The second problem could pertain to power, life support, a space suit's integrity (such as a leak), or perhaps just something annoying and difficult to resolve (such as a problem with the robotic laboratory) that is sufficient to distract the crew. In this case, assume that the problem is a malfunction in the coolant system of EVA2. Her suit quickly reaches 32C. She is perspiring and starting to breath more heavily. EVA2's personal agent (a program) informs her via a voice alert in her headset, as well as a red visual message in her HUD about the problem and suggests immediate return to the Hab. EVA1 also receives a message about the problem, and an audio beep and visual alarm appear at the IVA console in the Hab, with a verbal message broadcast on the loudspeakers.

In an attempt to get more information, the physician and one of the engineers in the Hab remotely command EVA2's suit to adjust a coolant subsystem parameter. By protocol this proposed change is automatically reported to EVA2 verbally by her personal agent, and she verbally affirms the command.

The crew realizes that they need to abort the EVA and report to the Hab crew that they will return to the spacecraft as soon as possible. EVA2 asks her personal agent for a walk back plan. A crewmember onboard the Hab who is responsible for monitoring the EVA most directly is verbally notified by his personal agent that a revised plan for the walk back path is available and asks him to approve it.

The EVA planning program has taken into account the metabolic condition of the two EVA astronauts, the status of their consumables, and status of their life support systems. In particular, the plan seeks to minimize time to return and exertion. The tradeoff requires a somewhat longer plan than a direct route, which would require walking up a steep hill. A simulation, which was automatically triggered by the replanning tool, further shows that the tool and sample carrier that the EVA crew is dragging along will not be stable along the new route and will cause an estimated 25% delay in arrival. The plan is represented on a map with a boxed alert that the crew should abandon the carrier in place for later retrieval. The responsible crewmember glances over the predicted route and safety statistics, which are displayed on a table computer, and approves the plan by touching the soft-control on the screen. (The plan is transmitted by email to Earth ground support.)

The EVA crew sees the planned route and warning on a computer tablet attached to carrier itself. They acknowledge the plan and warning by touching the screen, and simultaneously tell the habitat crew that they are beginning the walk back. (Alternatively, one of the EVA crew could have approved the plan verbally through his/her agent. About fifteen seconds have elapsed since EVA2 requested the walk back plan, most of which time was spent by the crewmember reviewing the diagram and associated charts.)

The responsible crewmember verbally instructs the robotic laboratory to interrupt its current science plan and to bring the tool and sample carrier back to the habitat. The robotic system verbally asks the crewmember whether returning as quickly as possible is important or should robotic battery power be conserved? The crewmember indicates timing is not important. Furthermore, the robotic laboratory has been serving as an over the horizon communication relay from the EVA crew to the Hab. Consequently, it must adjust its route and timing to maintain that functionality while the crew is walking back to the hab. To accomplish this goal, the robot's personal agent requests the EVA agent onboard the Hab for the EVA crew's planned route and predictive contour of their progress (where they are expected to be at each point of time during the traverse). The robot submits the power and communication constraints to the route planner program, which provides a navigation plan for the robotic laboratory.

EVA2 leads the walk back and receives an automated distance and bearing to the Hab every 5 minutes (according to her pre-assigned preference). At a certain point, when the habitat is out of sight, EVA2 asks her personal agent to increase the navigation feedback to every minute. Both EVA crewmembers receive alerts if they deviate from the planned path by more than 5 meters. They can verbally request directions to the Hab from any location at anytime. (All location and alert/status information is continually updated in the web pages logging the EVA on a server in the Hab; these pages are mirrored to Earth support as they are updated.)

The robotic laboratory is dynamically maintaining its communication relay function, adjusting its pace and replanning its route as necessary by monitoring the EVA crew's location.

Meanwhile, the physician onboard calls up current graphs of metabolic graphs and consumables. (The suit is maintained as the same atmospheric pressure and mixture as the Hab, so there will not be any recompression delay after doffing their suits.) The physician readies her instruments for a routine medical checkup.

Meanwhile, another crewmember in the Hab responsible for EVA system life support has a message waiting from an automated system describing the malfunction in EVA2's suit life support system with probable diagnoses based on past failure history of similar systems and data from suit tests prior to the EVA. A repair procedure on the onboard systems manual is referenced by a URL link in the message. The crewmember responds to this email with the text "Acknowledged." (A copy of the message with the acknowledgement is forwarded to engineering support on Earth.)

The EVA crew returns to the Hab without further incident. The robot follows two hours later, having automatically found the tool/sample carrier at the crew's turnaround point, coupled the carrier to itself, and followed a route plan suitable for its own efficiency (different from the crew's path).

4.4 Workflow Capabilities Illustrated in the Scenario

- Detection of off-nominal metabolic rate
  - Metabolic rate calculated by integrating biosensor data
  - Continuous monitoring of metabolic rate, compared to expected value for each crewmember based on prior EVAs
- Distributed alerting
  - Alert to EVA crewmember (Audio beep, visual message, and verbal message).
  - Broadcast of alert in the Spacecraft/habitat (audio beep, visual message on GUI, and broadcast)
  - Transmission of alert as Email to Earth flight surgeon PDAs
- Dynamic construction of EVA log
  - Automatically generated web pages on Hab computer, referencing an automatically created database with EVA records
  - Dynamic construction of a terrain map showing location of EVA crew and remaining planned route with workstations and schedule
  - Terrain map overlaid by predicted arrival times at waypoints, with predicted metabolic rates and consumables remaining for each EVA crewmember
  - Mirroring of EVA log to Earth ground support
- Configurable video feeds from EVA head cams to multiple display devices
- Detection having reached 90% threshold of life support consumables required for completing the EVA (with distributed alerts)

    - Continuous prediction of consumables required given route and work plan
    - Continuous monitoring of consumables, compared to flight rules
- Detection of malfunction in EVA suit coolant system (with distributed alerts)
    - Continuous monitoring of life support system behavior, compared to nominal performance
    - Alerting modalities (voice and visual message) determined by severity of problem and available methods (console in Hab vs. HUD)
- Remote commanding of EVA suit from Hab
    - Execution of protocol seeks verbal affirmation from EVA crewmember
- EVA route/navigation planner program takes into account the metabolic condition of the EVA astronauts, status of consumables and life support systems, and terrain relating time, effort, consumables, and prognosis of suit malfunction.
    - Subroutine automatically generates and compares alternative scenarios, including abandoning sample/instrument carrier to decrease effort
    - The plan charts predicted consumables and metabolic indicators (e.g., fatigue) during the EVA and percentages expected at time of return to habitat
    - The plan is presented on a GUI for the crewmember's approval with a verbal request and displayed message
    - Planner program transmits recommended route map and alert to EVA crew, displaying it on a tablet computer on the carrier
    - Crew acknowledges the plan and warning either by touch to GUI or verbally to agent
        - Verbal response will release the displayed message
- Verbal commanding of robotic system to bring tool/sample carrier to habitat
    - Interaction of robotic system with crew to determine task optimization criteria
    - Automated route planning, retrieval, navigation to habitat
        - Planner must integrate multiple constraints (retain communication relay function and minimize power)
        - Requires dynamic adjustment of robotic movements and replanning if crew route changes during the walk back
        - Continuous monitoring of crew position through a publish/subscribe mechanism
- Automated distance and bearing during walk back provided as verbal message to crewmembers (optional setting of frequency and visual cues on HUD)
    - Alert if deviation from planned route by threshold distance
    - Dynamic request for current distance and bearing to habitat
- Generation of EVA crews' current metabolic performance and consumables graphs in habitat
- Automated diagnosis (by habitat computer system) of ongoing malfunction in EVA suit life support system, with probable causes based on past failure history of similar systems and data from suit tests prior to the EVA
    - Reference to repair procedure in online manual
    - Crew member acknowledges receipt

- Diagnosis, suggested repair, and acknowledgement forwarded to Earth support engineers

4.5 Interoperability Functional Requirements for Accomplishing Workflow Capabilities

The vast majority of the capabilities mentioned above have been prototyped in experimental EVA systems, notably in NASA's Mobile Agents Project (Clancey et al. 2005, 2006, 2010b). In some respects, the flow of data, commands, and information can be anticipated and forms a kind of "workflow backbone." In general terms, the backbone can be outlined as follows:

{EVA Devices/Sensors/Instruments + Robotic Systems}
← *commands & telemetry* →
{Workflow Automation:
Data Exchange & Storage
+ Interpretation/Monitoring
+ Decision Support: Diagnosis, Planning}
← *goals/constraints, commands, information requests, telemetry* →
{Interactions with Crew:
Visual Media: Augmented Reality GUIs, Lights
+ Auditory Media: Beeps/Alarms, Spoken dialogue}
← *text/voice messages, video, databases, & web pages* →
{Interactions with Remote Support Teams}

In effect, the crew communicates with EVA systems in task-oriented terms, referring to plans, locations, and qualitative constraints (e.g., minimize power), but may also query about specific quantitative values (e.g., consumables remaining) and directly command or program subsystems (e.g., life support, robot). The workflow automation abstracts data (e.g., detecting off-nominal trends), converts goals/constraints to command sequences, and plans and dynamically controls subsystems in feedback relationships (e.g., a communication relay for a moving EVA crew). Communications with the remote support team are primarily summaries, but enabling on-demand "drill down" to recordings, telemetry, and detailed subsystem histories.

In more general terms, the EVA scenario illustrates two fundamental workflow capabilities whose absence in the International Space Station has been cited by NASA's space medicine specialists as reducing safety, accuracy, security, efficiency, and extensibility, while increasing DDT&E cost (footnote 4 above):

1. *Real-time data interpretation across subsystems.* For example, making inferences about the relations of bio-physiological data, suit consumables, life support system functionality, the EVA plan, locations of crew, robot, and habitat. These inferences usually involve abstracting data and comparing the current state to models of planned, predicted, and off-nominal behavior that must take into account the current context (e.g., as a simple example, is EVA1's breathing rate

abnormally high given the terrain, what she is carrying, and her spacesuit's performance?)

2. *Many-to-many communications among subsystems (no stove-piping)*. For example, throughout the scenario it is usually irrelevant where computing is occurring: an EVA Suit, Robotic Laboratory, or in the Hab. The workflow backbone enables any subsystem to potentially communicate telemetry and receive commands from any crewmember (through a variety of audio/visual devices) or computer program. (Section 7 provides an example of how this is accomplished.)

The operations of the robotic laboratory in the scenario particularly illustrate the kinds of workflow capabilities that are desirable and are provided by *semantic (high-level, task-oriented) interoperability.* The robot's workflow capabilities include:

- Can co-operate with people and other systems independently of medium available and/or chosen for communication
- Can receive new constraints, interrupt its current plan, and develop a new plan (possibly through an external planning program)
- Can flexibly execute a plan using dynamic feedback from multiple constraints
- Can accomplish high-level subtasks by accessing and adapting a procedural library (e.g., "retrieve sample/tool carrier"), which might be onboard the Hab
- Can derive additional information to create a plan by interpreting data from external sources (e.g., the current location of sample/tool carrier; the terrain; predicted location of crew during the next two hours to maintain its communication relay function)
- Can interact with people to get more information (e.g., crew preference on timing vs. power use)

The scenario also illustrates how safety and efficiency are enhanced by a system that the crew can *trust*. Through the combination of the EVA and Hab crews' situation awareness (via alerts, configurable displays, and ability to request evaluations of subsystem behavior) and the active monitoring and dynamic control of subsystems (e.g., life support, robotic laboratory), the crew gains trust in the overall exploration system—they know its current state, they can control it, and components are interacting automatically according to goals and current conditions.

In terms of interoperability, the crew trusts the capability of the robot, planning system, navigation assistant, life support monitor, etc. to "discover" the attributes of the situation and then synthesize the sequences required to accomplish the ongoing plan. This discovery process includes adapting plans according to changing goals (e.g., shift from EVA science to walk back to Hab) and capabilities of components (e.g., a failing life support system). The real-time data interpretation and many-to-many communications enable the crew to verify operations at any time—a key aspect of improving confidence in an autonomous system. They can relate plans to current states throughout the exploration system, both globally and by drilling down to get specific values and settings of any system through any convenient interface.

4.6 Discussion: Interoperability and Workflow Automation

This section provides a broad introduction to the nature of workflow automation in terms of what is meant by "work" and "information" in the context of exploration systems.

4.6.1 Work

In the context of LSS, *work* refers to the activities people and systems are performing on the surface, for example, scientifically surveying a chain of craters or experimenting with an ISRU. Work activities are oriented to the purpose of being at a particular location with certain logistic support and tools. In particular, scientific work, which we are focusing on in this report, goes beyond the life support, power, and mobility capabilities of the exploration system to consider, for example, what data is being gathered for what interpretative purposes. Although this may seem obvious, in large part exploration system design and mission planning usually begins with and emphasizes logistic requirements such as transportation, power, and life support, and then doesn't relate them to the functional requirements of interoperability for supporting the actual work.

4.6.2 Information

Part of the shift in mentality from thinking about engineering logistics to designing a work system is understanding the nature of information and in particular how it relates to data.

Data consist of actual subsystem measures, such as readings from sensors or other instruments, status or state information (e.g., whether a device is enabled; the size of available computer memory on a particular platform), or other "raw" formats, such as photographs, video files, voice communications, etc.

Information consists of contextually relevant interpretations—both of the state of exploration system and actions that are desirable—inferentially derived from data (e.g., the walk-back time from an astronaut's current location and whether the consumables will be sufficient at the astronaut's current metabolic rate).

Generally speaking, information is a distinction that is relevant to an agent's interests and/or goals (what Gregory Bateson, the 20th century anthropologist-cyberneticist, famously called, "a difference that makes a difference"). Here are some examples of information:

- Data contextually relevant to a recipient now. For example, a mobile EVA tool could superimpose a camera image with the names of craters. The selective display of data stored in a database, when associated with a perceptible feature constitutes information to the viewer.
- Abstraction of data by categorizing it (e.g., "high value"), an interpretation relative to some context.

- An alert that a condition is now true (e.g., an EVA waypoint has been reached), effectively relating time to goals, plans, resources, etc.
- Parameters for controlling a subsystem calculated by relating data (e.g., navigating to a waypoint on a safe path).

4.6.3 Investigative Work

Although mission planning today emphasizes meticulous scheduling of crew activities and calculation of resources, some work by its very nature involves open-ended investigation (aka "inquiry"). In particular, engineering maintenance and scientific exploration are activities that inherently involve the possibility that observations will affect the ongoing work by posing new tasks or suggesting alternative goals or priorities.

An investigation involves gathering data, assessing the situation, and deciding what actions to take (e.g., deploy or reset an instrument, repair equipment) and moving on to focus the investigation elsewhere. Investigative work often involves following procedures and being focused, but it is not itself strictly procedural (i.e., rotely enacted in a programmatic fashion). Investigations are inherently partly improvised, and so planning must leave time for consideration, and indeed provide means for frequent replanning (such as by recording why a step was included originally).

Frequently during an investigation, problems may be discovered, tasks may be better understood, and future plans may develop. "Carrying out an investigation" is an example of an *activity* (behavior in time): It is goal-directed, with a reactive observe-act loop. Actions involve getting data whose interpretation leads to deciding what other actions to take (e.g., getting more data, asking for assistance, deferring a problem, following a task/procedure). In short, the nature of investigative work requires flexible use of tools and inter-operation of hardware and software with people.

4.6.4 Workflow

Workflow involves processes of gathering, relating, translating, conveying, interpreting, organizing, and storing data and information, often by controlling and deploying tools (e.g., sensors or a robotic laboratory), within the context of particular traverses. Software that supports workflow by proactively carrying out work processes is called a *workflow automation system*. In practice workflow automation seeks to optimize a variety of constraints including safety and security, while assisting people to be more productive in accomplishing their work objectives. Note that human interaction and workflow are independent notions—workflow interactions (e.g., to assemble and interpret data for controlling subsystems) are still required when systems operation "autonomously."

As the scenario in Section 4.3 illustrates, basic workflow processes we expect during a surface EVA include:

1) *Creating an organized database* of the EVA activities, events that occurred, and work products (e.g., photographs and samples indexed on a terrain map)
2) *Alerting* crew members and remote support about what is occurring (e.g., sending an email to a remote science team; notifying biomedical support about a problem)

3) *Commanding subsystems* to change their behaviors in ways that promote the work (e.g., commanding a robotic system to use a particular instrument to carry something to a particular location; commanding a robotic system to inspect a pressurized rover for damage).

4.6.5 Summary and Appraisal

The scenario demonstrates how a workflow analysis can drive requirements for inter-operability, enabling diverse systems to work together. Put another way, architecture requirements are based on the *joint activities*—the work that components/subsystems and/or people are doing together and how the work flows among systems over time.

Workflow automation ranges from routine data transmission or information communications (e.g., passing documents and messages from one organization/role to another) to dynamically managed inter-operation of subsystems with feedback control. Along this dimension, the purpose of workflow automation shifts from a role of simple point-to-point relaying to more global orchestrating of co-operative interaction among people and systems. Through a roughly hierarchical design, different workflow processes can be assigned different roles that range from more narrow communication and transformation of data and commands, to data correlation and assembling a single command, to constructing sequences and controlling a feedback process.

This range of operations and orchestrating purview can be accomplished through different methods, though the simplest approaches will not be convenient or adequate for more complex co-operation among subsystems. Accordingly, some engineering approaches are suitable for coordinated control, but are not capable of the higher-order role of *orchestrating co-operative work.* In some respects, the problem of workflow automation is to glue together subsystems built using simpler integration methods, which are often physical and not reconfigurable. These subsystems may be moving in space (e.g., a robot or vehicle), temporarily brought together for a particular task (such as an EVA), and required to respond to unexpected combinations of events. Thus the problem of workflow automation is most generally that of flexibility, namely finding ways to adapt subsystems to: variable configurations and interfaces; communication methods, protocols, and security regimes; and goals.

The following section analyzes alternative systems engineering architectures for implementing interoperability functional requirements we have discussed in the scenario.

5 Architecture Trades for Implementation of Interoperability Functions

This section discusses the engineering challenges and tradeoffs of different software architectures for implementing interoperability functions. We clarify that we are focusing on higher-level interoperability and relate it to the concepts of an open architecture and middleware. We discuss how the shift from low-level to higher-level (workflow) interoperability involves shifts in mindset from *pair-wise integration* to *systemic inter-operation*, which is also the shift from the goal of integrating hardware to supporting work activities. Several candidate architectures are then discussed and compared with respect to the figures of merit for interoperability.

5.1 Low-level Interoperability: Network Connectivity and Data Exchange

A conventional basic engineering approach to interoperability is to standardize how devices/applications communicate, both in terms of connectivity and in data/commanding transmission. For example, in a multiagent system one may encapsulate applications and devices as "objects" that pass "messages." Communicating information requires a standard language for formulating and organizing electronic documents (e.g., "content templates") as well as an "envelope" that holds the content to be shared.

The global cell-phone network is an example of the failure to establish such standards: Different communication standards for voice format and delivery – CDMA, TDMA, GSM – are still used by regional carriers. The ability of callers to "interoperate" with other cell phone users depends on which "envelope" the carriers use to hold the voice data. Therefore, callers today from the US may need to purchase or bring a second phone to Europe if their phone uses a different standard (HIMSS-EHRVA 2005).

Some of the problems with the phone network exist in Space Station operations because of different security protocols and legacy systems requiring multiple networks, resulting in stove-piping of applications (Section 4.5). Section 5.5 also explains how adopting a "building-blocks" approach for ISS would fail to incorporate state-of-the-art technology that would enhance safety, timeliness, efficiency, etc. because it only focuses on connectivity and data exchange, not dynamic orchestration of operations.

5.2 Higher-Level Interoperability: Automating Information Creation and Exchange

Notice that the network phone example focuses on people interoperating with other people. In effect, low-level interoperability frames the communication system as *communicating boxes* that are exchanging data, that is, a *phone system*. Higher-level interoperability considers the purpose of the conversation, as an activity of *communicating people* who are creating and exchanging information for some contextually dependent purpose, that is, a *work system*.

For example, when communicating by voice, people are often attempting to explain something about their ongoing situation, which is facilitated by presenting something, which provides a common frame of reference, such as an image, audio recording, or graph (Kukla et al. 1992). Higher-level interoperability provides methods for people to understand each other—it takes into account what information is being communicated, the purpose, what triggered the communication, and other aspects of the context such as urgency. For example, in presenting information via a smart phone a person might be soliciting assistance to handle an urgent problem. The listener is oriented to determining what he or she should do.

In summary, workflow automation requires much more than providing connectivity on a network and a set of adapters for exchanging data between applications. Instead, workflow automation requires higher-level interoperability, which includes *proactive processes that automate some aspects of how information is created and transmitted*. For example, plug and play capabilities may be based on communication standards as in a publish-subscribe framework for automatically exchanging data between applications. Service-oriented architectures provide multiple adapters and communication protocols enabling applications to practically interact and be controlled by higher-level processes. The automated alerting in the scenario (Section 4.3) illustrates the functionality provided by a service-oriented architecture.

Higher-level interoperability may also include *methods for components that are joined on a network to automatically start working together.* Such dynamic reconfiguration is typically accomplished in a service-oriented architecture by a process of registering services (available functions or capabilities). Some applications may assume the role of an "executive" that actively monitors and invokes subsystems, a method used by model-based automation (e.g., Muscettola et al. 1998). Distributed control (i.e., lack of a centralized executive) is also possible if the architecture supports many-to-many communications among subsystems (Section 4.5), as in a multiagent system (MAS; Jennings, Sycara, & Wooldridge 1998; Wooldridge & Jennings 1995; Wooldridge 2002).

5.3 Relation to Open Architecture

Most architectures providing higher-level interoperability—including by definition those enabling dynamic reconfiguration (i.e., without recompilation of the entire system)—are de facto *open architectures*. That is, they provide methods for adding, upgrading and swapping components. Viewed over the longer term, across multiple missions, such flexibility satisfies the LSS "incremental buildup" objective by supporting upgrading and incorporating new elements (e.g., vehicles, robots, habitat modules). This allows 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).

If an operational interaction occurs between new and existing systems, then scalability may become a problem with an open architecture. Performance may decrease, and if adding new components requires changes to the interfaces of existing systems, the cost of modifications will increase. Consequently, methods have been developed based on standardized interfaces, involving layers of interaction, including how data and commands are physically transmitted and how they are packaged, so inherently the architecture enables any system to communicate with any other. This form of open architecture is analyzed in detail in the companion report, *LSS Software Open Architecture Study*.

5.4 Relation to Middleware

There are good arguments for why an interoperability implementation should include middleware, but this is not a necessity, and it is not definitional (though some may use the term to encompass even a service-oriented architecture).

We view middleware as providing only capabilities for low-level interoperability, such as the migration of point of control when a vehicle carrying an EVA crew becomes passive after docking with a habitat. The middleware is simply the means for a rather direct transfer of power and life support functions (Alena 2010).

Middleware per se is insufficient where a greater degree of flexibility is required during an operation. For example, subsystems may be designed at different times for different purposes, but they should be capable of goal-directed, adaptive co-operation. If a situation arises requiring an operation somewhat different than the baseline (or in the event of an off-nominal operation), the crew could trust the capability of the subsystems to "discover" the attributes of each other, and then synthesize the sequences required to accomplish the operation. The scenario in this report illustrates this capability in several ways, such as the planning service provided to the robotic laboratory by a habitat program. This kind of flexibility also enables high-level, goal-oriented commanding (e.g., the crew's directing the robotic laboratory to fetch the sample carrier, while still satisfying its communication relay function.).

Typically an exploration system providing higher-level interoperability includes a middleware framework. For example, some subsystems (e.g., ECLSS) might be integrated by middleware, while ECLSS itself could be inter-operating with other subsystems (e.g., a planning system) through other architectural methods, such as workflow agents.

5.5 Perspective: Shift from Connecting Components to an Operations Framework for Interoperability

Although we have emphasized the design of exploration systems consisting of separate physical units (e.g., habitat, rover, vehicle, instruments), the problem being addressed is more fundamentally a problem with software. Most computer programs aren't designed for interoperating. This is because programmers typically think in terms of "hardwiring" connections, rather than using a framework that enables the program's operation to be reconfigured by people at runtime. This mentality of connecting parts carries over into how people conceive of integrating physical systems.

5.5.1Example: What's wrong with today's smart phones

A familiar example of preconceived connectivity is found in apps written for today's smart phones—most are designed as "stove-pipe" processes. Consequently, users often find that they cannot access data or command from one app to another or integrate other devices. For example, one phone app converts voice to text, and it is easy to email the text. But to create a note, one must copy the text, shift to the note app, and then paste the text. Similarly, another app logs a terrain profile, associating photographs and video on a track overlaid on satellite images. This "trail recording" app uses APIs provided by the phone's operating system to enable interoperation of the GPS app, camera app, and mapping app (which retrieves images from the Internet). However, there is no way to record voice memos, even though the operating system includes such an app with an API. The problem is that the trail-recording app is not itself implemented as an open architecture (allowing a user to configure desired modules). More generally, the trail recording app and most of the hundreds of thousands of "smart phone" apps now available do not provide an API, so it is impossible to integrate them. The result is that for all of the power provided by these "location aware" devices and the "augmented reality" some provide, each app is closed to every other.

In part, the blame for the stove-piping of today's smart phone apps lies with the operating systems. Some may only provide a primitive form of multi-tasking (e.g., iOS4). But in general they do not provide *an interoperability programming framework* that promotes building apps that are dynamically reconfigurable and allow many-to-many communications with other apps. Thus, except for the operations preconceived by the designer, there is no interoperability and the user is continuously required to shift between apps through a central menu and limited to copy-paste.

By contrast, in higher-order interoperability required for workflow automation, the software architecture is open (enabling extensibility), and the operations of applications are actively coordinated by computer programs (e.g., "workflow agents") within the context of what people and subsystems are doing. Rather than point-to-point communications between applications/subsystems, higher-order processes can dynamically invoke a variety of applications/subsystems, constructing automatically the functional sequences required to control them by actively seeking data from other applications/subsystems and transforming it.

5.5.2 Replacing Pair-Wise Integration of Apps by a Flexible Systems Framework for Inter-Operation

The analysis of smart phone apps illustrates what happens when systems engineers view the problem of interoperability as preconceived, hardwired *integration* instead of cooperative interactions dynamically configured by someone in the operational context. The problem is that both the smart phone vendor who provides the operating system and most application developers view themselves as providing *apps*, rather than a *coherent system* in which the components know about each other and cooperate to help people accomplish their work efficiently, safely, securely, etc. within variable operational contexts.

When interoperability is not built-in, people must shift their attention from the work they are trying to do to low-level technical subtasks. A workflow system keeps attention on the high-level. For example, contrast the question, "How do I link my printer to my computer?" with a request spoken into a smart phone in a hotel room, "Provide a paper copy of this email." The first utterance asks how to get objects to communicate; it is a matter of pair-wise integration. The second, a request to accomplish a goal, *commands processes* to actively cooperate (e.g., discover what printers are available nearby, whether in the hotel or nearby store, and how to transmit the data and job order). The first asks for help in manually putting systems together. The second directly specifies and *delegates the work* the person wants accomplished.

In short, higher-level interoperability is not just about connectivity and communications, it is about how we get subsystems to act cooperatively in the service of ongoing task goals of the crew. In some cases, as in the printing example, these goals are explicitly about workflow. In other cases, the goals are in the background, involving comfort and health, conservation of resources, security, etc., such as the operation of ECLSS.

Consider the cooperation required for adaptively relating ECLSS operation and the crew's activity plan. The ECLSS could provide data to a planning system that monitors and analyzes how crew activities affect the power usage and performance of the ECLSS. The habitat's ECLSS could then be adjusted according to the crew's schedule and status, for example, preparing the habitat for occupation as the crew is returning from an EVA. In this respect, the habitat ECLSS would be acting cooperatively with the crew. Similarly, the ECLSS could provide information that the planning system would use for scheduling future activities.

From one perspective, the problem of higher-level interoperability is *getting different control systems to operate together*, with common goals, plans, and schedules, that is, to "co-operate." Co-operation means in general that subsystems adapt to each other. The shift from designing for coordination to designing for cooperation is a shift from hardwired physical sensors, effectors, and control systems to supporting the joint activity of people, hardware, and software systems. Most importantly, this joint activity must be dynamically configured according to changing circumstances.

In general, if a system is physically distributed, then a (heterogeneous) control process such as set of interacting workflow agents will be more appropriate than a centralized "control processor." A distributed control process is useful in any of the following circumstances:

- components can carry out operations independently (e.g., a robotic assistant, a suit life support system)
- subsystems can be dynamically configured (e.g., in preparation for or during different EVAs)
- communications are unreliable (e.g., because of terrain obstacles blocking the wireless network)
- operations of subsystems and people are not strictly synchronous and feedback may be time-delayed by distances (e.g., cooperation among Earth and interplanetary systems).

The problem of systems engineering for higher-level interoperability then becomes:

- *How to connect subsystems dynamically*, in order to support work activities
- *Providing data access to a device dynamically*, in order to create information presentations by consolidating/categorizing/fusing/analyzing data from multiple devices
- *Providing commanding access to a device dynamically*, in order to accomplish task goals by closed-loop controlling of multiple subsystems

In general, a linear, pair-wise integration perspective would truncate each of the above sentences before the word "dynamically": Connect subsystems, provide data access, and provide commanding access. In contrast, the higher-level interoperability perspective considers all of the components, hardware and software, as a coherent system that manipulates information and accomplishes work objectives. The next section more specifically examines current ways of talking about systems engineering for the International Space Station (ISS) from this critical perspective.

5.6 Combining design mindsets: Integrating Hardware that Supports Activities

Sometimes when designers adopt a systemic approach, they are still thinking in terms of integrating hardware rather than supporting work. A "both-and" perspective is useful: view "the system" as being both a *physical system* and a *work system*. To make this clear, several familiar ways of describing systems architectures are compared and related to the workflow perspective.

5.6.1 Viewing a System Architecture as Both Objects and Processes

In general, a physical perspective views a system as *objects*, which are described in terms of how they are connected and what they contain inside. A work perspective views a system as *processes*, which are described in terms of how they change over time, affect each other, provide services, and satisfy goals. Table provides a summary of these perspectives.

Table 1. Two ways of viewing a system architecture

OBJECTS	PROCESSES
Components (building blocks)	Flows (cooperating processes)
System-Based (applications/packages running on located workstations)	Activity-Based (on-demand services, available anywhere, anytime)
Centralized Hardware Appliance (e.g., Hub & Spoke, Bus)	Services (Distributed Agents)
Connectivity (Physical data & command transmission)	Proactive/Push Processing (Logical/Semantic/Event-driven)

In a distributed process-oriented architecture, the processes are active and interact in a network. This makes particular sense when the processes are running on platforms such as vehicles, space suits, and robots, which are mobile, independently operating, and in changing interactions, as noted above. Organizationally, this can be viewed as an "enterprise of enterprises," as promoted by the Network Centric Operations Industry Consortium's interoperability framework: "A network essentially is composed of acting nodes linked by relationships" (NCOIC 2009). We would add that the relationships might be dynamically changing.

The following subsections illustrate the object-oriented vs. process-oriented distinction in terms of the current medical systems onboard the Space Station and proposals of how to improve these systems.

5.6.2 Critique of the "Building Blocks" Approach

One common approach to systems engineering today is to adopt a common data management and communications architecture for creating "building blocks" that are configured to create multiple, interacting systems. This can be characterized, for example, as "One set of building blocks, many components" (footnote 4 above). The building blocks are components that are replicated and possibly combined differently for building subsystems into an integrated system (Figure 5.1).

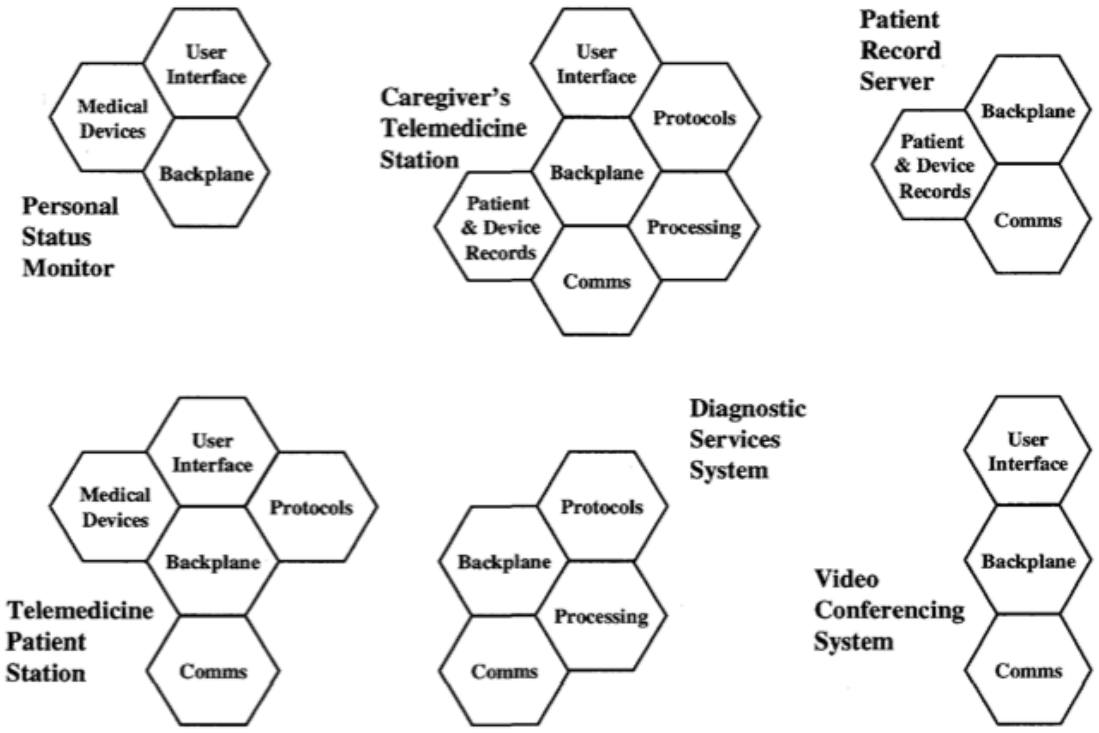

Figure 5.1. "Telemedicine Systems Created from a Common Set of Architectural Building Blocks," (Warren et al. 1999).

The building blocks in a telemedicine system might consist of communications, medical devices, a user interface, Records, a backplane (for integration), etc. The object-oriented perspective is continued at the next level to conceive of the telemedicine system in terms of "workstations" customized for different roles (e.g., Patient, Caregiver), services (e.g., archive records, diagnosis), and tools (e.g., video conferencing, personal status monitor).

The building blocks diagram is useful, but raises fundamental interoperability questions, such as: Is connectivity among building blocks and among stations/services/tools pair-wise and predetermined, or is it dynamically reconfigurable? With what effort? Does the system provide “anytime, anywhere” access to data and commanding, or does access require being physically co-located with a specific workstation or interface device?

One proposed architecture, which NASA Space Medicine specialists have been evaluating for adoption onboard ISS, does provide some dynamic “hot pluggable” capabilities through the “self-identifying” capability of a component and the registry function of the central Backplane of each subsystem (Figure 5.2).

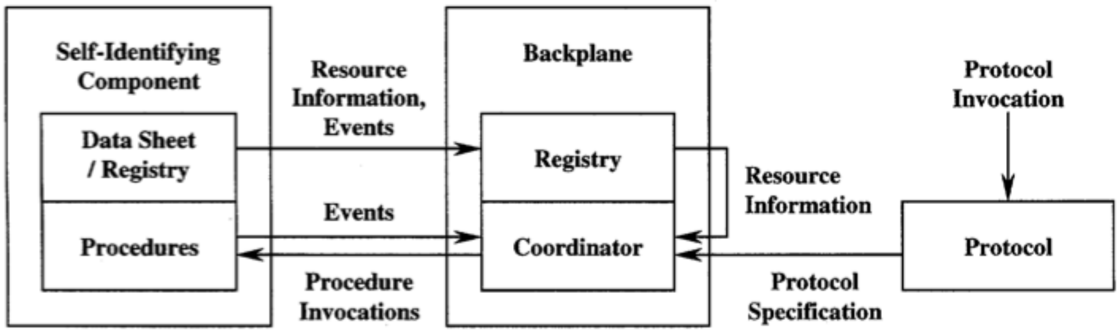

Figure 5.2. “Depiction of the general component interactions in the proposed architecture,” (Warren et al. 1999).

This example illustrates a kind of service-oriented architecture, in which a registered component will transmit events (to a “coordinator”) and can execute procedures (methods) on demand. The “Protocol” in this architecture is defined as the “resources, connections, and events necessary to accomplish a medical task.” A resource can be any computational process (e.g., medical device, data filter, GUI control), communication connection, or another protocol.

The conception of a “backplane” (or bus) for communication raises questions about flexibility, where control lies, and what automation is possible. In some respects, the “coordinator” can be viewed as an “agent” that follows a protocol to control the execution of methods (perhaps based on events specified in the protocol). Partly at issue is whether the protocols are flexible enough to provide alternative ways of accomplishing a task, such as when a component connected to the backplane is removed or disabled.

This example helps elucidate some of the requirements of workflow automation:

- Assembling, translating, and pushing data and commands according to service availability
- Translating information and request semantics to the data & command terminology of components
- Adapting behavior within task requirements to multiple, changing external constraints (i.e., context sensitivity)

It should now be clear that rather than the physical hardware metaphor of a “backplane” or “bus,” understanding higher-level interoperability requires a more process-oriented metaphor. For example, rather than talking about connected boxes (the hexagons in the example), we might talk about a network of "smart routers." Rather than viewing the problem as "integrating boxes," we might view the problem as creating locally independent subsystems that can act cooperatively by monitoring and proactively contributing to an overarching task.

From one perspective, the required shift is to provide more than a "common data and communication middleware" solution (footnote 4 above). A dedicated, hub-spoke box can provide telemetry and commanding to multiple devices that are co-located, as onboard ISS or in a surface habitat or vehicle. But referring to the scenario narrative (Section 4.3), what enables the robotic laboratory, habitat planning system, suit life support, etc. to act cooperatively with the crew? How do we make the medical devices, for example, aware of the crew's activities? How do we make people aware of the performance of these systems? This process view moves the architectural concept from a notion of centralized integration of physically connected systems to distributed systems—achieving interoperability through non-co-located resources and processes. Rather than a plug and socket type of registration to a controlling backbone, the services provided need to be more broadly available in a many-to-many mapping among applications and physical systems.

The following section presents and compares some candidate architectures for higher-level interoperability.

5.7 Candidate architectures for higher-level interoperability

In many respect, alternative interoperability architectures are based on different ways of viewing how computation is decomposed into subprocesses and how the output of these subprocesses are recomposed at runtime. One can view the progression of architectural concepts over the past 40 years as shifting the design perspective from structuring output (e.g., levels of abstraction), to structuring processes (e.g., layers of processes), to structuring flows (e.g., process networks). These are briefly described here as layered control architectures, service-oriented architectures, and workflow architectures. (See Moses [2010a] for a cogent summary of layered, tree, and network architectures relating organizational and computer system design.)

5.7.1 Layered Control Architectures

In the 1980s, artificial intelligence programming methods provided alternative methods for controlling multiple, asynchronous processes. Although the topic was sometimes called "distributed artificial intelligence," the processes were often located on one computer because of limited capability for process communication over networks at the time. The emphasis was writing complex computer programs as independent parts, rather than as one monolithic system. Two successful methods for dynamically orchestrating interactions between subprograms are described here: the Blackboard Architecture and the Multi-Tiered Architecture.

5.7.1.1 Blackboard Architecture

In a *blackboard architecture* (Nii 1986a, b), the overall computational product is a symbolic model of some situation or process in the world. In a typical application, the model might be an interpretation of a spoken utterance. The blackboard[5] was usually designed to have heterogeneous levels, corresponding for example to different model perspective (e.g., the grammatical parse of an utterance vs. its meaning). Subprograms, often called "knowledge sources," would asynchronously read what was posted on the blackboard and make contributions to the evolving model (e.g., using phonemic/morpheme information to construct words). They were often specialized by only reading and modifying particular parts of the model. An essential aspect of blackboard control structures is that posted information in the model might be uncertain and represent alternative interpretations. Frequently, the blackboard is a hybrid of a hierarchy (tree) and layered structure, such that related alternatives are compared and incorporated at a higher level, then becoming a hypothesized piece for the next layer above. Blackboard architectures are therefore useful for integrating sensor data; an early military application was to identify and track submarines in the Pacific Ocean by relating data from different sensors over time (Nii 1986a,b).

In summary, blackboard architectures are useful for assembling and relating information in a centralized place so it can be shared and improved by different processes that are assembling a global model with a layered structure. These processes interoperate in the sense that various combinations of processes are able to read and incorporate other processes' output.

5.7.1.2 Multi-Tiered Architecture

In a blackboard architecture the layers are structured objects constituting the output; in a *multi-tiered architecture* the layers correspond to processes that create these structured objects.

> In software engineering, multi-tier architecture (often referred to as n-tier architecture) is a client–server architecture in which the presentation, the application processing, and the data management are logically separate processes. For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. The most widespread use of multi-tier architecture is the three-tier architecture.[6]

In one formulation, the top tier corresponds to the presentation (display) level, the middle tier is the application or "business-logic" layer, and the lowest tier consists of data servers. Malin (1999) adopted the three-tier approach at NASA for developing a life support control system called 3T, consisting of a planner managing resources and life support products; a sequencer, a discrete event-driven controller; and a skill manager, which interfaces with hardware and continuous-control systems.

[5] The name "blackboard" is derived from a wall-mounted black slate board, used in American schools from about 1801 until about the 1970s, on which one wrote and drew with tube-shaped pieces of white chalk (gypsum).
[6] http://en.wikipedia.org/wiki/Multitier_architecture (1 Oct 2010).

The multi-tier perspective implicitly incorporates the blackboard architecture perspective that higher-level interoperability requires abstracting low-level data and commands into dynamically selected and composed sequences, which are planned and/or follow protocols. The multi-tier perspective is more general by not requiring a centrally posted model of the world and the output of subprocesses, and is thus amenable to a distributed-network implementation.

5.7.2 Distributed Network Control Architectures

In the layered control approach, the overall architecture conforms to the structure of the output (as a central blackboard or through layers of n-tier processing). In distributed network control architectures, control is not localized. In effect, the designer gives up the notion of controlling the overall system, but rather focuses on how people and systems co-operate dynamically when the configuration of subsystems is not necessarily known at design time or even fixed during operations. A combination of procedural and event-driven operation is therefore desirable. The emphasis is on enabling runtime reconfiguration of components with perhaps multiple methods for accomplishing work functions.

Such architectural flexibility is more appropriate for coordinating multiple semi-autonomous entities (people, organizations, and mechanisms), rather than for operating highly constrained components interacting in fixed ways (e.g., a scientific instrument). However, as once simple devices such as cameras and telephones become computer-controlled with multiple subsystems, are miniaturized and combined into pocket or wearable devices, it is becoming apparent that the methods described here are potentially applicable to any system; though again our emphasis in this report remains on work systems involving people and a wide range of automated systems and devices.

Today a networked control regime is most often called a *service-oriented architecture* (SOA):

> Service-orientation requires loose coupling of services with operating systems, and other technologies that underlie applications. SOA separates functions into distinct units, or services, which developers make accessible over a network in order to allow users to combine and reuse them in the production of applications. These services and their corresponding consumers communicate with each other by passing data in a well-defined, shared format, or by coordinating an activity between two or more services.[7]

A related definition is:

> A Service Oriented Architecture (SOA) is a software model in which the concept of a ‘service’ is an abstraction of a function used by an application. SOA logically decouples the service requester from the service provider by isolating the service

[7] http://en.wikipedia.org/wiki/Service-oriented_architecture (1 October 2010).

definition from a service implementation. (Freund & Niblett 2002)

The logical decoupling of service requester and provider in principle enables people using the system to configure components, rather than software engineers pre-determining what functions are available and how they invoke each other. An infrastructure or framework is provided that uses a form of mediation that acts as a kind of "match-maker." In effect, the location of services can be hidden, communication protocols may differ (e.g., "one-way, request/response, asynchronous, synchronous, and publish/subscribe type interactions"), and the message format or interface may differ (see for example, Freund & Niblett 2002). Conceptually, the designer shifts from thinking of standardizing and connecting processes to developing software routers that convert and transform communications so available subsystems can interoperate. Historically, the distribution of applications via the Internet drove the SOA concept, and the services are sometimes called "web services."

5.7.3 Example of Distributed Control: Multiagent Workflow Systems

A multiagent workflow system is a particular kind of service-oriented architecture, in which mediating processes, called *workflow agents*, embody the "business logic." Because the agents communicate, it is amenable to a physically distributed network implementation. For example, in the EVA scenario (Section 4), workflow agents can be used to orchestrate the flow of data, information, and commands among sensors/instruments, audio/visual devices, automated subsystems (e.g., life support system, robotic laboratory), and web services (e.g., a remote database, a planner). The "logic" here is similar to the layers in the multi-tiered architecture, by logically relating data, information, and control through processes of abstraction and model-based inference.

Software agents (or "actors") usually operate asynchronously, but communicate with each other and subsystems to carry out delegated actions. Such a framework of interacting agents is called a "multiagent system" (MAS) (Wooldridge, 2002). Workflow automation fundamentally changes how system interactions occur by shifting the problem from how systems *interface with each other* (usually at the data and functional level in the language of each system) to how systems *interface with workflow agents*, which mediate among applications to accomplish task-oriented goals. In effect, what are usually conceived as "pipes" are now pro-active processes (agents) that transform and direct how data, commands, control, and information flow through the system.

In a multiagent workflow system, the agents are akin to "knowledge sources" in the Blackboard architecture, and more generally can be viewed as evolved from the knowledge-based programming approach. The agents are usually functional experts that invoke each other as required to handle subproblems, which themselves have designated specialists to handle them. Each agent encapsulates what is colloquially called the "knowledge" of how to process a request (incoming data or control command) to make it able to be processed by other agents or subsystems (e.g., restructuring information into a message that an email system can process). More strictly, this "knowledge" involves creating and/or manipulating models of the world situation and processes the workflow system implements. Furthermore, each agent manages the I/O relations required for its own functionality and any agent can invoke any other as it requires, allowing it to be used for multiple purposes in different configurations. Programmers can in turn modify their agent's abilities to process data/commands as new capabilities are introduced (e.g., a navigation agent, rather than merely specifying a heading and distance, is modified to invoke a planning system that evaluates alternative routes based on cost/benefit tradeoffs). Internally the agents are structured as "case statements" (switches) so new capabilities do not interfere with existing ones.

Section 8 provides a particular implementation of a multiagent workflow system. A comparison of multiagent systems appears in Clancey et al. (2008). For another perspective on multiagent software engineering methods, addressing software product lines, but not interoperability, see Peña et al. (2006).

5.8 Summary: Trade comparison relative to figures of merit

Table 2 compares the candidate software architectures we have discussed for interoperability, relative to the figures of merit (Section 3.2). Alternatives are scored on a relative scale of 1 (worst) to 10 (best). The values are relative, attempting to discriminate inherent characteristics from the effort required to achieve the figure of merit requirement. For example, ratings for Distributed Network Control below 10 reflect that quality will depend on the programming of the relevant agents. In particular, security could be impaired by network vulnerabilities that are common to the other architectures (while timeliness might be much easier to reliably provide). Data management is more tractable through the agent's functional design, but agents might become overloaded by simultaneous requests and high volumes—these aspects must be considered and designed into the overall system as for other architectures. Similarly, crew efficiency is by design enhanced by an agent system, but prototypes must be used experimentally to discover what people will need and want to do. Incremental buildup and DDT&E were rated 10 because these are inherent in the architecture.

The evaluation of current ISS systems is adapted from the recent NASA space medicine evaluations of the current hardware and software infrastructure (footnote 4 above).

Table 2. Comparison of candidate software architectures relative to figures of merit driving interoperability.

Figures of Merit / Candidate Architecture	Security & Safety (accuracy, timeliness)	Data Management (volume & integration)	Incremental Buildup (scalability, extensibility)	Operational Efficiency/ Productivity (crew time)	DDT&E Cost	Σ
Stove-piped Systems (e.g., ISS in 2010)	2 (lacks automation; impairs safety: can't support entire crew simultaneously)	2 (lacks automation; not time-synchronized; limited data reduction)	1 (limited multi-device connectivity; lack of data exchange methods; difficult to automate)	1 (manual work, cognitive burden; inefficient air-to-ground bandwidth)	1 (redundant, proprietary, and mixed technologies)	7
Layered Control (e.g., 3T)	3 (monolithic, not amenable to international access)	7 (designed for model-based control)	5 (hardwired services; no exchanges)	5 (designed for automation)	4 (ad hoc data exchange)	24
Building Blocks (e.g., next generation CHeCS[8])	8 (customized workstations, variable security possible)	5 (requires standards)	8 (designed for reuse)	7 (designed for work activities)	5 (must revise existing apps for standards)	33
Distributed Network Control (e.g., Multiagent Workflow)	8 (designed for adaptability; optimal for time-delayed communication & multiple network topologies)	8 (designed for creating & distributing information)	10 (designed for extensibility, plug & play; mobile systems)	9 (adaptively optimizes workflow; maintains crew focus on task)	10 (maximizes code reuse, multiple uses)	45

6 Recommendations: Software Architectures for System of Systems Inter-Operability

This section summarizes the analysis and conclusions and relates the interoperability recommendations to the proposed Constellation C3I systems architecture. The possibility of a "clean slate" approach is discussed, as well as enduring challenges for interoperability.

[8] Proposed next generation crew health care system (CHeCS) for ISS space medicine operations with "Dedicated Communications Gateway/Hub & Archival Storage Server."

6.1 Summary of Analysis and Conclusions

This report has reviewed the nature of software interoperability, distinguishing low-level and higher-level capabilities, and compared broad, well-known approaches (layered control, building blocks, and distributed network control) to exploration systems figures of merit (safety, security, data management, incremental buildup, productivity, and cost). Distributed network control was found to be the best approach by all measures. In particular, a multiagent implementation, which facilitates workflow automation, was recommended.

Different perspectives are useful for designing a complex integrated, distributed system, including networking, inter-process communication, interfaces, and workflow. In practice, a distributed network architecture will in some respects involve building blocks and functional layering. The distributed network perspective is more embracing by showing how multiple design perspectives can be related into a reconfigurable and robust operational system of systems.

The workflow perspective brings together two design perspectives for building a complex system:

1) *Human-Centered Design: Systems exist for human purposes and inevitably involve human interaction.* In particular, people are the "endpoints," that is, they are being served by the system components. Although, for example, habitat system monitoring may be automated, eventually people will be interested to know that the habitat system is functioning correctly, as well as that the monitoring is working correctly, and they will need to respond to anomalies.

2) *Task-Oriented Design: Interoperability of sensors/devices and automated processes involves model-based abstraction and inference to support both human decision making and enable automated processing.* For example, working safely on EVA requires relating consumables, metabolic performance, and workload—whether this is done by a person or assisted by a system.

In general, workflow automation involves associating, packaging, and communicating both discrete and continuous data streams; the purpose may be archival or for ongoing operations, by either people or automated systems. For example, EVA photographs are more efficiently organized for later interpretation if the database automatically pulls together data about location, time, camera source, and the context of the EVA activity. This is accomplished by synchronizing and often reformulating data from diverse sources at runtime, creating new information representations (e.g., web pages, text summaries).

Interoperability shifts the systems engineering notion of "integration" from connectivity and forwarding telemetry and commands between subsystems to *orchestrating operations in a system of systems to create and manage information.* Interoperability is concerned with enabling higher-order processes to control and hence coordinate actions while allowing incremental buildup of subsystems within the ongoing context of human tasks (work).

Workflow generally involves an interactive, ongoing communication of requests to act and to provide information. In general, communications among subsystems are many-to-many, with each component receiving and giving requests and information from several other components. For example, during an EVA subsystems for data and commanding of life support, navigation and planning, and science data collection often need to dynamically coordinate their behaviors. In general, these systems are distributed, communicating without direct, wired connections, and in general some of the components are mobile. In practical implementations, middleware provides a medium for communicating data and commands among devices and applications, while workflow agents orchestrate the flows.

The kind of interoperability required within a system of systems depends on whether and how particular components interact:

- *Low-Level Interoperability (Middleware)*—for systems whose components operate independently and/or interact only in fixed ways (e.g., functional hierarchy) and tend to be of fixed types (e.g., plug-ins).
    - Applicable to self-contained subsystems & frameworks (e.g., ECLSS, a PC OS, web-browser), which nevertheless may be open architectures by virtue of enabling addin-applications to run within their closed framework.
    - Typically, added functionality is independently accessed (e.g., new menu) or independently used (e.g., new GUI, software tool).

- *Higher-Level Interoperability (Workflow Automation)*—for systems whose components are distributed, heterogeneous (i.e., not designed to a single standard), and are intended to interact with other software and/or hardware (e.g., a planner subsystem; a database; an Email program; a printer).
    - Components are typically added with the intent that they will be able to interact with other components.
    - The Mobile Agents Architecture provides a framework for building such systems

We concluded from analyzing the EVA emergency scenario that a Lunar Surface System with workflow automation (Section 4.5) would significantly improve safety and productivity by enabling a diversity of distributed subsystems to interact flexibly (e.g., habitat, pressurized vehicle, spacesuit, robot, instruments). These workflow functions include real-time, closed-loop coordination of multiple subsystems; consolidating, filtering, and abstracting data; creating and forwarding relevant information; and assembling data from multiple subsystems to construct commands for devices/applications.

Besides making inferences and constructing command sequences, workflow agents may present information to people, expressing it in verbal speech or an audible tone, representing and storing information in dynamically constructed web pages, and/or transmitting information by email. Some information may consist of goals suggested to the astronauts (e.g. “Begin walking back to the habitat now”) or conveyed to “autonomous systems” (e.g., “Scout, go to Astronaut 2”). Such inferences require that agents have access to and/or maintain models of the current exploration system (e.g., the location of Astronaut 2; what Astronaut 1 is doing) and the plans and procedures by which the work is being carried out.

In summary, sophisticated workflow functions include the heuristic guidance of people, robotic systems, and software agents to facilitate cooperative work (e.g., parceling tasks among people and robotic systems and delegating information processing to software agents). Such systems use available information to plan, alert, guide, record, control the ongoing flow of information and commanding to assist or directly orchestrate inter-operation.

Such higher-order services, when provided by a “workflow backbone,” are not device or subsystem dependent. Because they operate at the task level of EVAs, they are inherently extendible and may be repurposed for very different exploration system configurations (e.g., see the companion report, *LSS Software Open Architecture Study*). Such an architecture is particularly amenable to incremental buildup—including substituting/upgrading or adding subsystems and enhancing workflow capabilities.

6.2 Interoperability Recommendations Relative to C3I Architecture

A key claim of this report is that the proposed Constellation C3I architecture should be elaborated to include information exchange services in the form of workflow automation to facilitate and proactively initiate interactions between “element-specific data systems” (Figure 6.1).

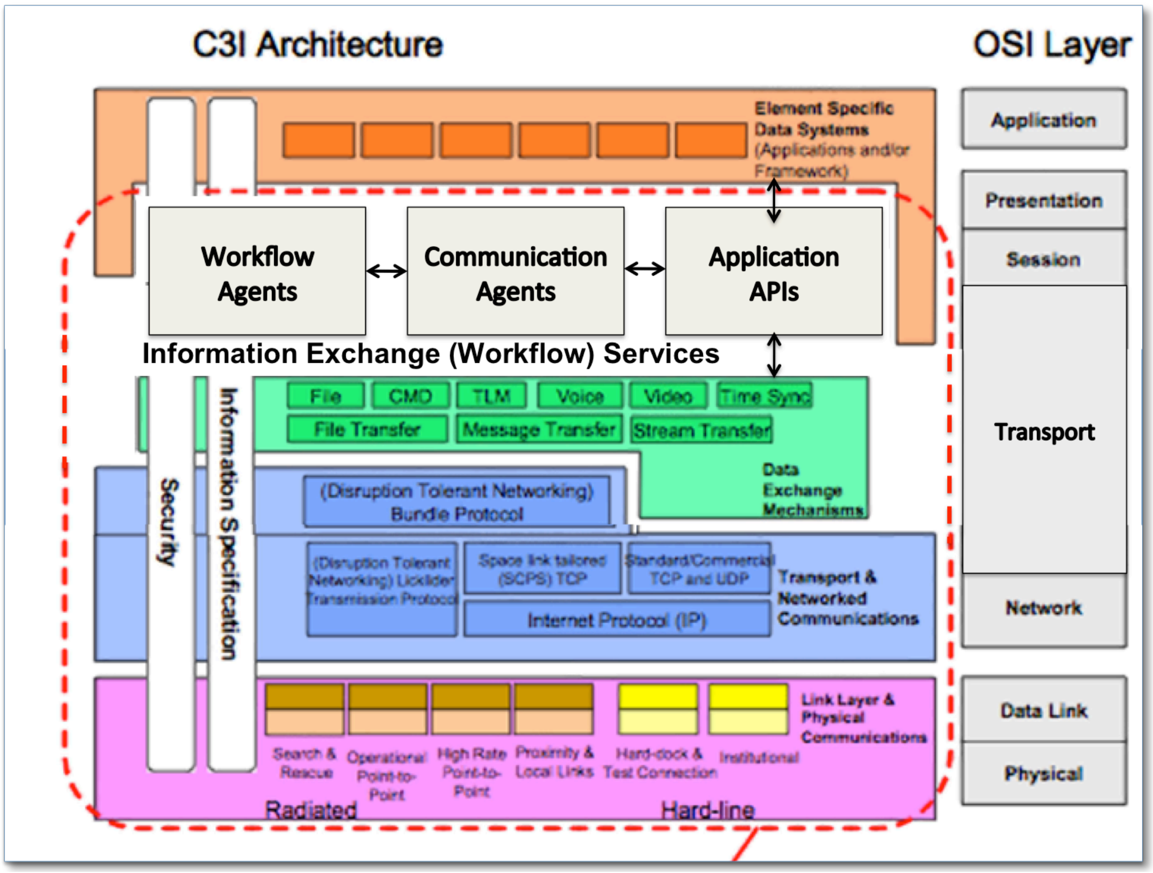
C3I Architecture
OSI Layer
Element Specific Data Systems (Applications and/or Framework)
Application
Presentation
Session
Transport
Network
Data Link
Physical
Workflow Agents
Communication Agents
Application APIs
Information Exchange (Workflow) Services
Security
Information Specification
File
CMD
TLM
Voice
Video
Time Sync
File Transfer
Message Transfer
Stream Transfer
Data Exchange Mechanisms
(Disruption Tolerant Networking) Bundle Protocol
(Disruption Tolerant Networking) Licklider Transmission Protocol
Space link tailored (SCPS) TCP
Standard/Commercial TCP and UDP
Internet Protocol (IP)
Transport & Networked Communications
Link Layer & Physical Communications
Search & Rescue
Operational Point-to-Point
High Rate Point-to-Point
Proximity & Local Links
Hard-dock & Test Connection
Institutional
Radiated
Hard-line

Figure 6.1. C3I Architecture interoperability layers and their mapping to the OSI Reference Model, including Information Exchange (Workflow) Services layer. Transport layer is elaborated to automate flow of information between element-specific data systems, using data exchange services and transport/networked communications; See **Figure 8.2** for details and examples. (Figure adapted from CxP 70022-1, Figure 1.5-2; dashed red line represents scope of C3I Interoperability Specifications).

Information exchange services are logically part of the Transport layer in the Open System Interconnection (OSI) Reference Model[9] that C3I adapts.[10]

In terms of the C3I Architecture and its specifications for interoperability, this report makes the following recommendations (refer to Section 8 for details):

1) *The C3I Architecture "data exchange mechanisms" should include active, information exchange services to enable goal-oriented automation within the exploration system* (Figure 6.1).

[9] For example, see *http://en.wikipedia.org/wiki/OSI_model*.

[10] One might also depict the Workflow region above the Applications. It is placed below because the C3I architecture is described as enabling applications to communicate. We suggest having that communication go through an information creation/exchange layer, not directly from app to app. In particular, depicting the WF backbone below the apps eliminates the interpretation that apps are communicating directly through the data exchange layer.

- In the specific architecture described here, these information exchange services constitute a *workflow system*, which consists of software agents that mediate between "element-specific data systems" to gather data, create information, and initiate feedback relationships that accomplish work tasks.
- Alternatively, this recommended elaboration may be viewed as inserting an Information Exchange Services layer as a distinct part of the Transport layer of the C3I Architecture, or as constituting a kind of integration framework that is part of the Application layer.

2) *The data exchange layer of C3I should specify information exchange services (e.g., secondary APIs called "communication agents") that mediate between the model-based language of the task domain, and the functional methods and data objects of the application programming interfaces (APIs) provided by "element-specific data systems."*

    - The language of the task domain includes the names and properties of objects (e.g., instruments, vehicles, robots), locations (both geographic names and targets, e.g., "waypoint 5"), human participants, work products (e.g., photographs, samples), and the work plan (e.g., an activity's purpose, location, duration, participants, equipment).

    - Communication agents, specialized for each integrated hardware and software component, translate from the language of the task domain to the component's exposed application methods and data objects, effectively making components into agents that provide task-oriented services. That is, components can provide information and not just data, and they can be configured to carry out task-oriented goals that dynamically relate to other components in the exploration system versus carrying out isolated commands.

3) *Preferably, people should be able to make requests for information and actions to occur in natural language (i.e., as speech acts) using voice commanding.*

    - However, other media such as visual display text and menus can be used to formulate requests in the language of the task domain, and other interfaces (e.g., tones and buttons) can be used for communicating information and commands that are formally translated to and from speech acts by software agents.

It is important to remember that with respect to LSS, what the C3I Architecture labels as "element-specific data systems" includes devices (instruments, tools), robots, and vehicles that include software for externally controlling their operation and/or providing access to internal data (e.g., status, measurements). For example, a camera that can be configured and operated, as well as provide access to stored photographs, via a cable or wireless connection is an example of an "application" in this context.

The distinction between data and information (Section 4.6.2) provides another way of understanding the importance of information exchange services. Workflow automation focuses on information exchange and therefore directly exercises the objective that C3I interoperability is intended to address (see also definitions in Section 3.1):[11]

> Central to the ARCHITECTURE is the use of a framework or data exchange services mechanism that uses publish and subscribe communications over an information bus. The framework allows individual application components to "plug-n-play" by supporting interface standards and facilitating the interactions with the system and between applications. (CxP 70022-1, p. 12).

In this respect, the term "data system" is not general enough to cover the computational systems typically included in an exploration system. Alternatively, one could interpret the original C3I Architecture as relating specifically to software, and imagine a layer above of physical systems in which the software is operating (e.g., a robot, life support system, scientific instrument). Therefore, "applications" include operating systems for controlling complex machines (e.g., VxWorks used on MER).

A key problem in designing for application interactions is to develop a limited number of interfaces and preferably a common way of exchanging data and information. The notion of a "common way" or standard often suggests that all components can communicate in a single language, but this is both technically unnecessary and makes incremental buildup with international partners much more difficult (if not impossible). Nevertheless, managing information and real-time control requires some efficient way for communications to occur without continuous translation back and forth through different standards.

In the suggested agent-based workflow architecture, this efficiency is attained by a backbone of workflow agents, providing basic services, which remain relatively consistent among system configurations. The agents all speak the same language and then communicate with subsystems through customized translators called *Communication Agents*. For a complete discussion of a workflow backbone and communication agents, see Section 8.

[11] The dashed line in CxP 70022-1 Figure 1.5-2 appears to exclude application interactions and so it might be argued that information exchange, and in particular workflow automation, exists a level above the diagram. However, CxP 70022-1 defines interoperability as concerning exchange and use of *information* between systems, so we have placed information exchange within the transport layer, but straddling the applications, as data exchange services were shown in the original diagram. From another perspective, information exchange services constitute a framework for communication between applications, and so might be represented within the Application layer. However, in Mishkin's presentation in Rader (2008) "framework" refers to a publish and subscribe model that includes a toolkit of building blocks and shared components for developing applications. The agent-based interoperability framework described here is a method for integrating components, rather than consisting of parts from which new applications are constructed.

6.3 Legacy Issues Hindering Interoperability — Why not a Clean Slate?

The legacy systems of the International Space Station provide a useful example for considering to what extent a "clean slate" is desirable, rather than designing around legacy systems. For example, most onboard medical devices lack wireless connectivity, onboard storage, and/or adequate APIs—why not simply replace them by redesigned hardware? The simple answer is that this is extremely expensive and has few short or long-term advantages—several well-understood integration technologies are already in practice, all handling legacy systems through some form of middleware (Table 2).

Another argument is that there will always be "legacy" systems that are incompatible with technology advances, particularly if components are supplied by international vendors, which is already common. Just as ISS space medicine technology was provided by different vendors using different technology over several decades, it is unrealistic to assume that even with a clean slate approach, a single standard for interoperability adopted today could be entirely suitable when astronauts are exploring the surface of an asteroid or Mars.

Rather than consider a clean slate, a more useful question is whether the kinds of problems faced by space medicine systems integrators today are solved in a better way by adopting a software interoperability architecture providing distributed network control?

As noted in the discussion of Table 2, a distributed network "process" framework would be more useful for future automation. CHeCS is based on a building-blocks approach, perhaps because the integration focus is on co-located, stationary hardware systems (e.g., treadmills, sensors, computer networks). We argued (Section 5.6) that designing the higher-level interoperability of workflow automation involves shifting from focusing on the physical system to focusing on the activities of the people. Analysis of both routine and problem scenarios can help shift focus from the basic logistics of accessing data and remote commanding, to developing a concept of operations for facilitating cooperation among people and machines. This design methodology is described in Section 8.

6.4 Enduring Challenges for Interoperability

In conclusion, many of the broad operational requirements addressed by interoperability are likely to persist in future exploration systems:

- Performance for real-time systems (including response to human actions)
- Adaptive response for plug and play operation (e.g., dealing with loss of assets such as loss of a robot during ongoing work)
- Flexible levels of subsystem commanding (e.g., robot API providing access to its sensors and effectors)
- Security across diverse network/transport protocols
- Semantic interoperability (relating different ontologies)
- Scalability, particularly to dozens or hundreds of subsystems

Simply put, interoperability involves managing a diversity of technologies in a complex, already functioning system. In part, diversity stems from technological advances that make today's state-of-the-art systems "legacies" in a few years. Although we might say that incremental buildup is desirable because we cannot afford to build everything at once, it is also inevitable because we cannot afford to replace everything at once when technology advances. We will always need to deal with legacy systems that generate, represent, and store data in different ways, as well as upgrades that seek to use existing data and control legacy systems in new ways. Inevitably, operating systems and computational platforms will continue to evolve. Rather than a "clean slate" approach, we recommend a framework that is designed for reusability, extensibility, and scalability.

But besides the inevitable existence of legacy systems, diversity stems from the diverse origins of hardware and software. The separation of people into functional organizations imposes further challenges for interoperability.

For example, some of the difficulties faced by biomedical engineers arise because mission operations and space medicine are funded independently, and so they develop systems independently. They naturally adopt different standards as well because operations and medicine have different criticality and security constraints, leading to the use of multiple networks and communication protocols. For example, in ISS operations today ground security prevents the human factors support team, which provides regular news and entertainment files to the ISS crew, from accessing mission operations servers to store the files they want to upload to the ISS. Thus stove-piping these organizations' responsibilities obstructs the workflow—a problem resolved in Mission Control by using a distributed network control framework (Clancey et al. 2008).

Mission requirements aside, managing large groups of people and designing complex systems both lead to a hierarchical structure, which then is reflected in functional stove-piping of requirements and designs. For example, during FY10 consideration of new technologies, a "Human Health, Life Support and Habitation" sub-team developed a hierarchy of four "subsystems" in a Human Research Program roadmap: ECLSS and Habitation Systems; EVA Systems; Human Health and Performance; and Environmental Monitoring and Safety. How do we investigate these topics without breaking them into parts? But how can we design for interoperability of the sort exemplified in an EVA emergency scenario (Section 4), if we view the Hab ECLSS, EVA life support, and Crew Health as inherently independent? The recommended approach is to adopt a programming framework that facilitates designing exploration systems from multiple perspectives—layered and networked, building blocks and services, physical systems and human activities (Moses, 2010a).

The "coherent system" perspective, particularly in the context of a complex system of systems operating in a hazardous space environment, raises issues of standards and governance in an ecology of inter-operating subsystems In this respect, the figures of merit (Section 3.2) organize the requirements that must be satisfied systematically and drive the architectural support for ensuring that the subsystems respect standards for security, safety, timeliness, extensibility, efficiency, and cost.

How a community would verify and legislate standards for communicating and interacting is on the fore-front of today's smart phone operating system technology. Leading providers are taking opposite approaches—Apple's iOS "sandboxing" strictly limits interoperability outside the Apple-provided services; Google Android's inter-process framework enables customized networks of apps. Methods for preventing deadlock, runaway applications, malicious commanding, and so on will begin with how these matters are resolved by conventional computer operating systems.

Moreover, for designed and for the most part closed systems like LSS, the inter-operating components will be selected and their services certified before mission operations. Our concern will then shift to validating overall system usefulness in inherently unknown environments where subsystem failures will occur. Accordingly, in healthcare and air traffic management where people and automation share responsibility for operations safety, a key research topic is designing resilience, such as checks and balances to cope with failure; NASA's exploration enterprise will do well to follow and contribute to these research communities in the coming decades.

7 Appendix: C3I Interoperability Objectives and Methods

This section quotes excerpts from "Constellation Program Command, Control, Communication, and Information (C3I) Interoperability Standards Book, Volume 1: Interoperability Specification" (CxP 70022-01) relevant to the definition and requirements for an LSS open architecture and interoperability. Excerpts are sequential with page numbers from CxP 70022-01.

> Interoperability is defined (based on IEEE 90) as the ability for two or more SYSTEMS to exchange information and to use the information that has been exchanged. For Constellation, C3I interoperability includes the common mechanisms for communicating between SYSTEMS, as well as the common information structure and language necessary for SYSTEMS to perform appropriate COMMAND and control functions using the information exchanged. (CxP 70022-01, p. 7)
>
> The Constellation Program consists of multiple SYSTEMS. The SYSTEMS used by the crew consist of the CEV, the Lunar Surface Access Module (LSAM), the Suit Systems (EVA) and Flight Crew Equipment. In the future, the Mars Transfer Vehicle (MTV) and the Mars Descent Ascent Vehicle (DAV) will be added to the active program. The launch vehicles include the Crew Launch Vehicle (CLV) and the Cargo Launch Vehicle (CaLV). The common support SYSTEMS consist of the Mission Operations and Ground Operations SYSTEMS. Finally the destination surface SYSTEMS consist of the habitat, surface mobility, power systems, robotic SYSTEMS and resource utilization. Significant external interfaces to the Program include the Communication and Tracking (C&T) Networks and the International Space Station (ISS).

This specification is applicable to all Constellation SYSTEMS. However, not all SYSTEMS require the same functionality due to their role in the larger Constellation ARCHITECTURE. For this reason, the specification is organized such that functional interfaces are applied consistently across many different SYSTEMS. This approach allows Constellation to 1) minimize unique interface definition / implementation and 2) maximize the potential for interoperability. (CxP 70022-01, p. 7)

The C3I ARCHITECTURE is based on common communications links and protocols, common COMMAND and TELEMETRY formats, interoperable voice and motion imagery protocols, and common information definitions. The C3I ARCHITECTURE leverages industry advances in network and data systems to address key challenges of the Constellation Program including simultaneous COMMAND and control of multiple systems, use of diverse communications environments, AUTONOMOUS operations support, cost constraints, and configurations and operations concepts which will continue to change over a period of many years. To address these challenges, the C3I ARCHITECTURE defines a loosely coupled, interoperable SYSTEM-of- SYSTEMS ARCHITECTURE based on the use of open, standards-based interfaces and switched/routed end-to-end communications networks. (CxP 70022-01, p. 11)

Layering is used both to isolate functionality and to aggregate commonly-used functions. Each layer provides a set of services to the layer above it by using protocols to exchange information with its peer layer as well as services from the layer below. So long as the interfaces to the layers above and below are maintained, the technology and protocols used to implement the services at a particular layer can be changed with minimal disruption and at relatively low cost. This will allow the Constellation C3I ARCHITECTURE to grow and evolve gracefully as technology changes. (CxP 70022-01, p. 11)

Central to the ARCHITECTURE is the use of a framework or data exchange services mechanism that uses publish and subscribe communications over an information bus. The framework allows individual application components to “plug-n-play” by supporting interface standards and facilitating the interactions with the system and between applications. Above the network protocols are the data exchange protocols necessary for command and control, including COMMANDS, TELEMETRY, time synchronization, voice, MOTION IMAGERY, and file transfer. (CxP 70022-01, p. 12)

The Data Exchange Layer defines a set of common data exchange mechanisms which can be used to simplify application development and testing, and improve application performance and interoperability. These “mechanisms” provide Constellation SYSTEMS a standard approach to share COMMAND, control and end-user information with one or more other systems. (CxP 70022-01, p. 14)

This section covers near-term requirements for the protocols necessary to exchange data between SYSTEMS. Specifically, this section discusses IP-based protocols for routine voice exchange, motion imagery transfer, data exchange messages for command and telemetry, file transfer and time exchange. Additional data exchange mechanisms and data types include static host table updates, SNMP traffic, key management messages, and traffic management control files. (CxP 70022-01, p. 57)

Constellation SYSTEMS shall use a common data exchange message format as defined in Appendix C of CxP 70022 Constellation COMMAND, Control, Communication, and Information (C3I) Interoperability Standards Book, Volume 5: Data Exchange Protocol Specification. [C3I-135]
***Rationale**: Common data exchange message formatting supports interoperability between SYSTEMS.* (CxP 70022-01, p. 61)

8 Appendix: Example Implementation—Mobile Agents Architecture

This section describes the Mobile Agent Architecture, focusing on the nature of workflow interoperability. DDT&E statistics using MAA for developing exploration systems are provided in the companion report, *LSS Software Open Architecture Study* (Clancey et al. 2010b), summarized in (Clancey et al. 2011).

The MAA provides an interoperability programming framework by the use of a common messaging scheme, allowing any hardware or software component to communicate with any other. Interoperability is enabled by proactive computer programs called "agents," which manipulate data and commands among components. They execute on-demand or through scheduled or event-driven procedures. MAA agents run on distributed, usually mobile computer platforms that get data from and control subsystems via APIs provided by each component.

To clarify some of the terminology, the MAA essentially provides an agent language, messaging representation and handling scheme. In the dual-API architecture, a distinction is drawn between the workflow agents that communicate in the language of the task (e.g., using terminology and actions of an EVA plan) and the communication agents that translate the task language to the implementation language used by the APIs of subsystems. Data exchange, transmission, etc. is handled by a middleware framework called the Collaborative Infrastructure (Clancey et al. 2010a; Alena 2010). It includes a distributed directory service, support for point-to-point message passing (transport service), and publish/subscribe method of message passing (data distribution service). There is currently no persistence of messages (store and forward), nor does it currently support streaming.

As an example instantiation, consider the general architecture of OCAMS shown in Figure 8.1. Five software systems (the NOMAD email system, Electronic Flight Note system, SWRDFSH (an FTP program between ground and ISS computers), and Microsoft Word and Excel are shown with their API. These APIs are "wrapped" by dedicated communication agents (Comm Agent) implemented within the Collaborative Infrastructure SOA, thus through this "dual-API" scheme in which the Comm Agents serve as a secondary API for the application, the five applications become "services." The workflow agents are services, too, in this framework. Notice that the apps don't communicate directly, but interoperate through their comm agents.

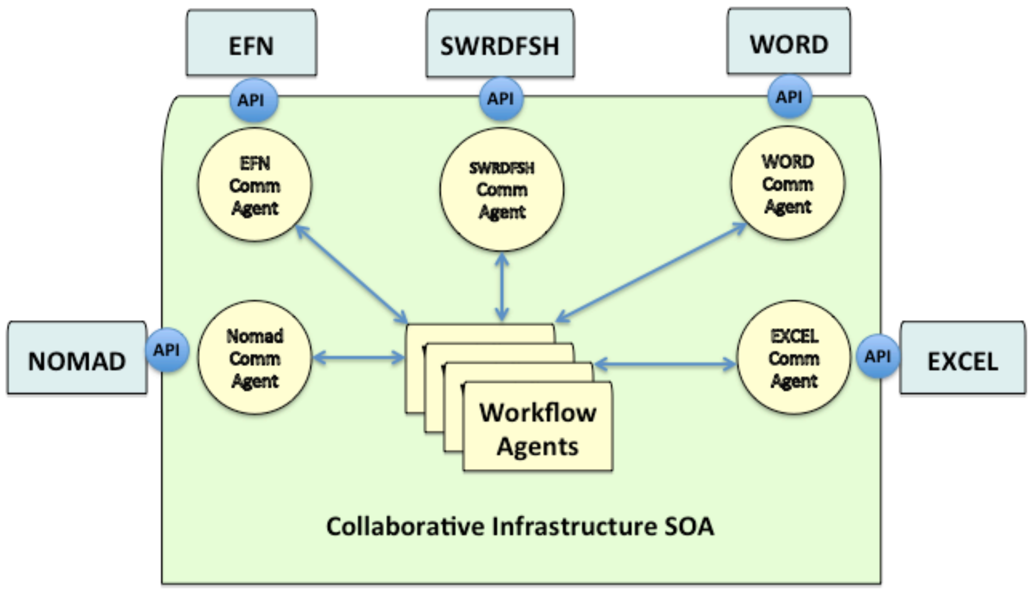

Figure 8.1. OCA Management System (OCAMS) basic architecture. OCAMS is a *workflow application* implemented as *agents in a service-oriented architecture* that acting together enable information exchange and subsystem interoperability.

Applications of the MAA have involved a set of exploration systems that enabled interoperability of robots, instruments (biosensors, cameras, geophone device), databases, and software (metabolic rate interpretation, Email, Microsoft Office). The "interface" consisted of natural language spoken alerts through headphones and loudspeakers, beeps (e.g., to confirm commands), visual alerts on a standard computer GUI, a web-browser science database integrating plans and data (called ScienceOrganizer), and commanding through either GUI buttons or natural language voice commands through headphones or computer microphones.

Because of the variety of interfaces, the ability to get and receive information or give commands has to be decoupled from the processes that provide and process crew requests. In effect this means that workflow capabilities must inter-operate with multiple interfaces. In particular, every new component (e.g., a temperature probe) must at least indirectly interoperate with auditory, visual, and spoken interfaces. Consequently, the command functionality is not associated with the interface, but rather the agent architecture provides a mapping of multiple interfaces to multiple programs that respond to requests. Very simply, each interface has a dedicated Communication Agent, which forwards requests to the appropriate Workflow Agents for processing. This functionality with examples of interoperability required is illustrated by the EVA scenario (Section 4).

More specifically, for the case of voice commanding, processing occurs in three major steps:

1) translating to and from natural language speech (i.e., a speech processing application);

2) breaking down or reassembling the information/commanding request by reformulating it in structured objects and relations that represent the data and commanding functions provided by different subsystems (or other workflow agents);
3) translating the request to the language of the subsystem APIs (comm agents).

By comparison, a conventional GUI more directly maps data and commands to the visible menus and displays, usually through code associated with the interface itself. Subsequent sections provide details of how MAA is applied to create exploration systems with interoperating components.

8.1 Logical Design of a Workflow System

Table 3 provides a systems design perspective of the MAA, showing how planned EVA scenarios (level I) are formalized as capabilities people have for getting information and controlling systems through a command language (level II), which in turn is processed by workflow agents interact with each other to convey, transform, and assemble messages (level III), which when suitably packaged are conveyed as structured objects to special communication agents that translate the request into the parameterized language of an external system API (level IV), which then further processes the system function to retrieve data and/or control an external hardware or software system (level V), which then looks up stored data and/or takes an action modifying its state according to the command (level VI).

Table 3. Exploration System Design Partitions in the Mobile Agents Architecture

Level	System Layer	MAA Constructs	Representational Units
I	Scenario	EVA Plan: Activity Schedule & Route	World model shared by agents
II	Human-System Interactive Functional Capabilities, including I/F and Services	Command Language	Grammar with Variables and Constants
III	Workflow	Brahms Workflow Agents	Tell/Ask Message Exchange
IV	Workflow Interface	Communication Agents	Mapping: Structured Objects <–> Functions
V	Component Interface	API	Functional I/F
VI	Component Subsystems	External System	Bus-level Control & OS

This layer representation is logical in the sense that it conceptually relates different aspects of a system’s design. In particular, a scenario of course is not a component; in practice, MA field scenarios consisted of: 1) network configurations for hardware and software included in a prototype exploration system (e.g., robot, email, camera, databases), 2) A "script" describing EVA goals and plans including routes with waypoints and scheduled activities to be performed at different sites using certain equipment, and 3) Desired workflow functions (e.g., schedule alerts, automatic creation of science database and web pages, email notifications to remote scientists). System design and development was strongly organized by the EVA script, however voice commanding functions enabled modifying the plan during the EVA, effectively using the pre-formalized plan as a menu of activities that could be skipped, reordered, carried out at different locations, and extended in time. In the runtime system, the scenario is formalized internally as an EVA Plan, referenced by the workflow agents.

The logical layer description also glosses the design of the workflow system itself:

- *The workflow agents were physically distributed on multiple computer platforms communicating wirelessly.* Each platform consisted of a network of agents, communicating with one or more component subsystems connected to that platform with cables or using Bluetooth.

- *The agents themselves organized into groups in the Brahms language*, so for example the "personal assistant" agent of each astronaut inherits common properties and behaviors from the "personal assistant" group; thus, AstroOnePA and AstroTwoPA are members of the AstronautPA group, with different configuration files causing AstroOnePA to be instantiated (created) in Astronaut 1's computer and AstroTwoPA to be created on Astronaut 2's computer.

- *The method of interacting between an API and a CA is not fixed, but varies according to the constraints imposed by different components.* For example, the RIALIST voice commanding component (Dowding et al. 2006), which is implemented in Prolog, uses the Prolog-based OAA (Open Agent Architecture by SRI) and provides a Java API; consequently, the RIALIST CA[12] (which straddles RIALIST and Brahms workflow agents) uses programming libraries from both the OAA and Brahms Java. In contrast, the ERA CA uses CORBA to interact with the ERA API, which is written in C++.

Figure 8.2 shows how parts of the exploration system logically interact as a software architecture, visualizing the different levels shown in Table 3 as they might occur in a typical configuration involving a robotic system. Circles represent hardware or software, except for the central circle which represents a network of workflow agent systems running on multiple platforms. Workflow agents serve as the backbone connecting subsystems. Connectivity is shown as an overlap of circles, in which two programs interact: On the agent side "communication agents" (CA) receive TELL/ASK requests from workflow agents; they then translate and transform these requests into suitable parameterized functions provided by the APIs of external systems. These API functions retrieve data from or control their associated systems (e.g., download images from a camera).

[12] This is called the Dialog Agent in the code, but RIALIST CA is used here to make explicit association with RIALIST.

The workflow agents communicate in terms of "speech acts" that express goals—what applications are supposed to do or the information they are requested to provide—in terms of the objects, locations, and tasks of the work domain (e.g., "Tell me when my metabolic rate is abnormally high given the work I am doing"; "Boudreau (robot), take a picture of Astronaut 2."). Speech acts are of two types: *Conveying information* (Inform) and *Requesting information or an action* (Request). These speech acts may also be referred to as "Tell" and "Ask." For example, an agent may subscribe to another agent's services by a Request; an agent might publish a service by communicating it using an Inform act.

A special agent associated with each hardware and software system being integrated (the Communication Agent) mediates between the language of the work domain in which speech acts are expressed, and the functional/data language of the component system (i.e., its internal methods and data objects). Thus for example, an astronaut can give the command "Scout, follow me" and the Scout rover's operating system will begin using data provided by a location device worn by the astronaut to control the rover's motion—effectively sustaining a dynamic goal-directed feedback interaction between the rover and the moving astronaut (or more precisely, for example, the GPS device worn by the astronaut).

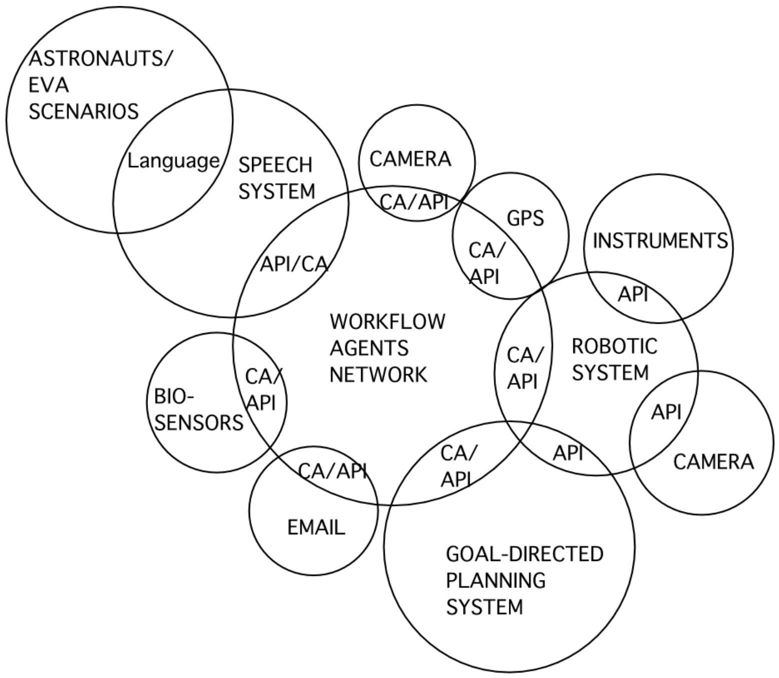

Figure 8.2. Logical Design of Typical Agent-Based Systems Integration Architecture, relating scenarios of system behaviors to networked hardware and software systems. Circles represent people and hardware/software components, with interaction made possible for the people by a language and for the components through the interactions of an associated API and "communication agent" (CA). See text for further explanation and compare to **Table 3** and **Figure 8.3**.

Some external systems, such as robots, pass on commands to their own peripheral systems (e.g., cameras, instruments) through internal APIs (e.g., in this manner SCOUT configures and commands its camera to take a picture). Software subsystems can also interact with both workflow agents and robotic systems. For example, in the Collaborative Decision Systems (CDS) configuration (Pederesen et al. 2006), a goal-directed planning system operated by people in the simulated surface habitat controls a robotic system. Workflow agents can also access the planning system (e.g., to provide data to the human operator which is useful for planning & controlling the robot's actions).

Conceptually, an overall system configuration is designed from the EVA scenario requirements (level I), from which a voice commanding language is derived (level II), which is in fact part of the speech processing program, itself just another external software subsystem. (Note that the language used for voice commanding could also be used for a visual, text-based interface or buttons on the spacesuit representing the commands.) During the EVA, astronauts' voice commands are relayed by the speech system to the workflow agents, which consolidates, transforms, and translates required data through other agents. Similarly coming from the other direction, any external system can provide data that workflow agents transform into expressions for the speech system to say over loudspeakers or to the individual astronauts, for example, to answer an astronaut's question directed at a particular robot, "ERA1, who are you following?"

Typically an agent designed to monitor and provide alerts about either desired or anomalous conditions (e.g., a life support resource is low) will receive data regularly from an external software system (through the API/CA interfaces) or will poll external devices or other data processing systems. When the agent detects the condition of interest (e.g., an exceeded threshold configured for a particular EVA), the agent will construct an appropriate expression to be uttered by the speech processing system. This design has been used for providing alerts about astronaut health, system health, and managing the EVA plan (e.g., "Five minutes remain at this workstation").

Figure 8.3 shows the same typical configuration, with the level numbers from Table 3. Starting from the "outside," all connected software and hardware systems are designated as level VI, represented as shaded circles. (In the diagram the speech software system and automated subsystem(s) are distinguished by name because they play special roles in the logical design of the exploration system, explained further subsequently.) The "external" hardware and software systems each have an API (level V), which interacts with a communication agent (level IV). This is represented in the diagram as an overlap between the external system and the workflow agent network.

Let's return to the distinguished external subsystems, the speech and automated systems. The astronauts are interacting with the speech system (which could be a natural language text system on a visual display or in highly restricted form, a device with buttons corresponding to commands). In the logical design, the astronauts' behaviors (during an EVA or while working in a habitat or spacecraft) correspond to the scenarios that represent the desired and expected work processes. In practice, the scenarios are formalized as plans on a timeline with location information (e.g., an EVA route and schedule of activities) and available to one or more workflow agents.

In Figure 8.3 the robotic system has been generalized to be any "automated system," by which is mean a system that itself controls one or more subsystems. An example is a life support system in a pressurized suit in the POGO 2007 configuration. In practice, an operating system in the automated system translates and reconfigures commands coming through the API in order to appropriately control its subsystems (e.g., to control the motors of SCOUT so it moves to a designated waypoint). In this manner, the overall exploration system design is hierarchical, though coordination originates through the commands of people and goal-oriented subsystems (if any). As shown in the figure, these commands may be conveyed indirectly through workflow agents (level III) or directly through software interacting with automated systems.

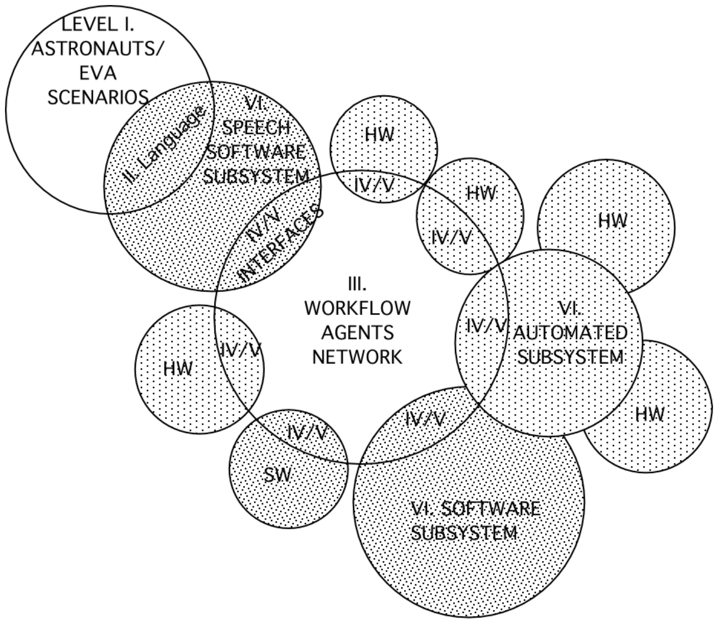

Figure 8.3. Typical exploration system configuration (Figure 8.2) related to the generic design layers (Table 3). The diagram illustrates the generality of the architecture: Workflow agents integrate hardware and software subsystems. An "automated subsystem" (VI) is itself an integrated system, and its subsystems can be controlled through the workflow agents. Other distinguished subsystems are interfaces with people, exemplified here by a speech system that provides language level, goal-oriented access to information and system-wide control.

The management of automation to adhere to flight system rules lies outside the scope of the agent-based systems integration architecture; the architecture provides a means for interaction and communication among all subsystems by which control protocols may be implemented. However, it must be stressed that all of the prototype configurations, from which this architecture was derived, the scenarios (level I), language (level II), and workflow agents (level III) were designed such that the overall system would operate deterministically, always under human control. Specifically, voice commands were designed to facilitate accomplishment of the work plan, directly control all external systems, including configuring and disable/enabling every automated system (e.g., modify alerts, tell a robot to stop, change the EVA plan).

One of the MAA configurations (CDS05; Clancey et al. 2010b; 2011) enabled an astronaut to control a rover through a goal-oriented planning system. In certain respects, from the perspective of the astronaut natural language commanding is goal-oriented: An astronaut requests information required for making decisions necessary for carrying out his or her tasks. For example, appraising the capabilities and progress of automated systems and remaining resources enables choosing among alternative methods for carrying out a task. Voice commanding is also used to directly instruct devices and robots, etc., to carry out subtasks or to assist. For example "download photographs" means not just to transfer them from the camera to a drive but to send them to a database agent and properly log the associated location, time, and contextual data, deploying devices, carrying tools. The agents must interact among themselves to gather and represent the necessary data. An important research opportunity is to experiment further with goal-oriented planning systems that interact directly with automated systems and indirectly with other components of the exploration system through workflow agents.

8.2 Natural Language Contrasted with Data and Command Displays

In exploration systems developed with MAA, people use natural language to make requests for information and give commands that require agents to pull data from multiple sources and transform it (e.g., "What has been the maximum amp load in the habitat this morning?" "Scout, take a picture of this site.") In using voice commanding (as opposed to standard computer displays) the system designers were able to focus on what astronauts on EVA or in a habitat actually wanted to know or do at given moments. That is, the system was designed to integrate data and provide information rather than simply display pieces that people would be required to find in different windows and put together themselves.

It referring to the modality of an interface, we might list the alternatives as visual, auditory, tactile, and gestural. However, although voice commanding uses auditory signals, it refers to a *representational level*, not just a sensory modality. In this respect, we should be wary of calling voice commanding an "interface" as if it may be directly compared to visual displays. Rather the most direct comparison is to NL text input and presentations on a display. The display is the modality; NL is the representational expression of the interface.

Natural language is a unique way of specifying what you want to know or have accomplished:

- NL interactions are expressed in terms of information and action; most visual displays are simply unorganized collections of words that people must assemble to get answers or issue commands.
- Voice commanding integrates data to answer questions useful for understanding situations, decision making, and controlling systems.

- Easily made available anywhere anytime (except extremely noisy environments) via wireless headset—minimal local apparatus, no visible displays, keyboards, or even touch screens required.

The MAA provides this functionality by enabling arbitrary system integration through software agents that transmit, translate, and assemble data on demand for storage, alerting, and control. For example, to execute "Scout follow me" agents continuously retrieve and transform location parameters for commanding the robot so it tracks the astronaut.

The design constraints imposed by NL voice commanding have several advantages for surface systems architecture research: 1) Ensures that the functionalities developed are those that are useful in (simulated) mission contexts (vs. asking, what is all of the data available and how can we organize it on the screen), and 2) Leads to exploring in different scientific and practical contexts the advantages of voice commanding, that is, getting information and giving commands without having to manipulate an interface.

Natural language is of course how people normally communicate with each other. A hypothesis of this research is that NL would be preferred for communicating with software and hardware systems. NL is not necessarily the only method of interaction we want to use; however, the advantages listed above made it an obvious candidate for managing workflow during EVAs.

The most salient examples of EVA "workflow" are task-oriented, focusing on specific flow of data, commands, and work products (e.g., a route plan); these tasks are part of an activity in performing an ongoing role (e.g., being the navigator in a pressurized vehicle). However, not all workflow (and perhaps not most) involves a frequent or continuous flow of data/commands/work products to and from people. Information may be provided as an alert to astronauts when an anomaly occurs, requested by people to troubleshoot an apparent problem (e.g., "Boudreau, who are you following?"), requested as a periodic status check (e.g., "What is astronaut-2's current activity?"), occur at stages through the work process (e.g., "call this waypoint 4"), and may involve changes to the ongoing work plan (e.g., "extend the current activity by 10 minutes").

By virtue of the nature of voice commanding during activity, interpretation and proper response often requires context-sensitive integration of information and/or construction of appropriate commands. For example, the automated biomedical monitoring system (POGO 2007) prompts astronauts to adjust their EVA plan according to ongoing measurements of metabolic rate and resources, which are directly affected by their task-oriented behaviors. That is, context sensitivity is an inherent requirement in order to be responsive to every astronaut request for information or command. Additional flexibility is gained by allowing indexical constructs that reference time, location, and object, such as "since 11 am," "here," "your location," "the last voice note." Resolving the meaning of such utterances often require the agents to request and relate data from different subsystems (e.g., location records, activity plan, science database). In effect, astronaut requests are broken down and passed along to other agents as messages, eventually arriving at an API of a hardware or software system that can provide the data or carry out the request.

8.3 On-Demand Workflow Automation Contrasted with Command Sequencing

It is worth noting how voice commanding relates to spacecraft command sequences. In fully automated "command sequences" uplinked to programmed robotic systems (such as the Mars Exploration Rovers), consideration of timing and system interactions are critical, such as managing power and device-specific operating constraints (e.g., warming up an instrument before it is used). For systems like MER, these individual and overall system constraints are anticipated and managed during the operations planning and uplink preparation process on Earth. By contrast in surface exploration scenarios, voice commands are on-demand requests that may occur at any time, rather than a batch uplinked and executed in the future (though of course any voice command may specify a future time).

Moses (2010b) summarized this distinction between control required by a device with components whose individual operation is mutually constrained and a system of multiple, semi-autonomous entities as follows: "Networks can be even more flexible than layered system and certainly tree-structured ones, but are often not easily controlled…. The [network] approach emphasizes cooperation in organizations through lateral alignment of groups at the same layer." This lateral alignment in MAA is represented by the workflow backbone.

So for example, with respect to MER, an astronaut could be standing near MER (or in a pressurized rover or habitat) telling the robotic system what to do (e.g., to take a microphotograph, do spectral imaging, go to a waypoint), and receive the data back in due course through the agent network. (Indeed, the measurement might have been programmed by an agent to answer a more abstract question posed by the astronaut, and the resulting data would be interpreted accordingly.). In this case, MER's communication agent (at level IV) would be responsible for managing simultaneous operations, very likely invoking planning and sequence testing systems (similar to those used for programming MER uplinks) that would be real-time, computational services available as agents in the exploration system on the surface.

In the MAA system configurations developed to date the various components are either operating independently (e.g., GPS, biosensors, web recorder) or only activated by direct control (e.g., a camera). Conflicting controls are possible (e.g., an astronaut is driving a vehicle when someone else commands it to go to a waypoint automatically); these must be anticipated and modes or override protocols provided.

In general workflow backbone agents are acting mechanically to forward, translate, etc. data and command requests. Relevant control issues include efficiency (e.g., an agent that becomes a bottleneck) and scale (e.g., bandwidth available is exceeded forcing queues that result in processing delays). Agents generating alerts require more careful design and some degree of situation awareness to be relevant (e.g., not repeating the same alarm in a distracting manner) and to avoid interference (e.g., interrupting people while they are speaking to each other). Such operational pragmatics are discovered through experiments with prototype systems in authentic work situations, such as those conducted in the Mobile Agents project at MDRS (2002-06).

8.4 On-Demand Workflow Automation Contrasted with Conventional Office Workflow Systems

The term "workflow automation" has been chosen to characterize the overall computational service provided by the MAA. This term was chosen because some aspects of the communications among agents and applications are similar to COTS workflow automation/management systems (for examples and characterization see Clancey et al. 2008). These tools focus primarily on tracking the execution of tasks by people and automated systems with respect to a relatively fixed workflow process structure, such as processing a customer's purchase order. One can map out in advance all of the players, subsystems, and process steps including conditional flows (e.g., handling backorders).

Certain aspects of designing an MAA system have a fixed, well-defined nature like workflow tools, but the overall nature of the work system being created through agent-based integration and the services provided to people are different. Indeed, the MAA could be applied in office environments to improve an employee's situation awareness and capability to manage work, exemplified by OCAMS's orchestrating file transfer between the ISS crew and ground support (Clancey et al. 2008).

Here is a summary of what MAA workflow automation accomplishes and how it provides services different from conventional office workflow tools:

- Services adapt to the changing context in which a human is working (e.g., in a habitat vs. walking on EVA)
- Agents are designed to cope with missing or additional services according to the current configuration
- Information is not merely transmitted in pipes connecting applications, but rather actively interpreted, assembled, transformed, packaged, etc.—in this respect the automation is not merely in *flowing* work from one subsystem to another as in conventional office systems, but in *doing* aspects of the work.
- The communication agents provide a way to integrate arbitrary hardware and software systems so they are "agentified," speaking the same language; most office workflow tools focus on software integration and are conventionally designed to interact with the subsystem APIs directly, rather than bringing them to a common level of communication.
- Voice commanding replaces the vast majority of conventional typing and interface manipulations that office systems require.

8.5 Agent Communication in the Workflow System

The ability to communicate data and commands among components lies at the heart of the agent-based workflow approach. The method for achieving this communication using the Mobile Agents Architecture evolved through experience, usually to make integration with existing systems easier and more robust. Different methods continued to be used, according to the constraints imposed by different components.

In particular, the integration of the speech recognition and generation system (i.e., RIALIST, a research version of the now commercially available NUANCE system, Dragon Naturally Speaking) with the Brahms agents demonstrates how the MAA accommodates integration with other open architectures using agents. This appendix briefly describes how the method for inter-agent communications was improved over time and specifically how SpeechActs were used.

Originally, in Desert-RATS 2002 agents were running on only one platform, so distributed communications were not necessary. Simulated EVAs at the Mars Desert Research Station (MDRS) in Utah in 2003 involved three platforms (two astronaut computers and the habitat communicator's computer, HabCom); "proxy agents" were introduced for communicating with an agent on another platform (Figure 8.4). The proxy agent would be configured with the network address of the corresponding agent.

This method proved awkward, requiring a proliferation of proxy agents with fixed configurations, and was replaced at MDRS 2004 by a centralized directory service using KaOS and a program called the Location Manager. KAoS initially used CORBA as its transport layer; that is, Brahms workflow agents on different platforms communicated over CORBA. In later versions of KAoS this transport layer was replaced with TCP/IP. In the CI, which replaced KAoS in the MAA for the OCAMS implementation, CORBA data structures are used for ComActs, but TCP/IP is used for remote agent communications.

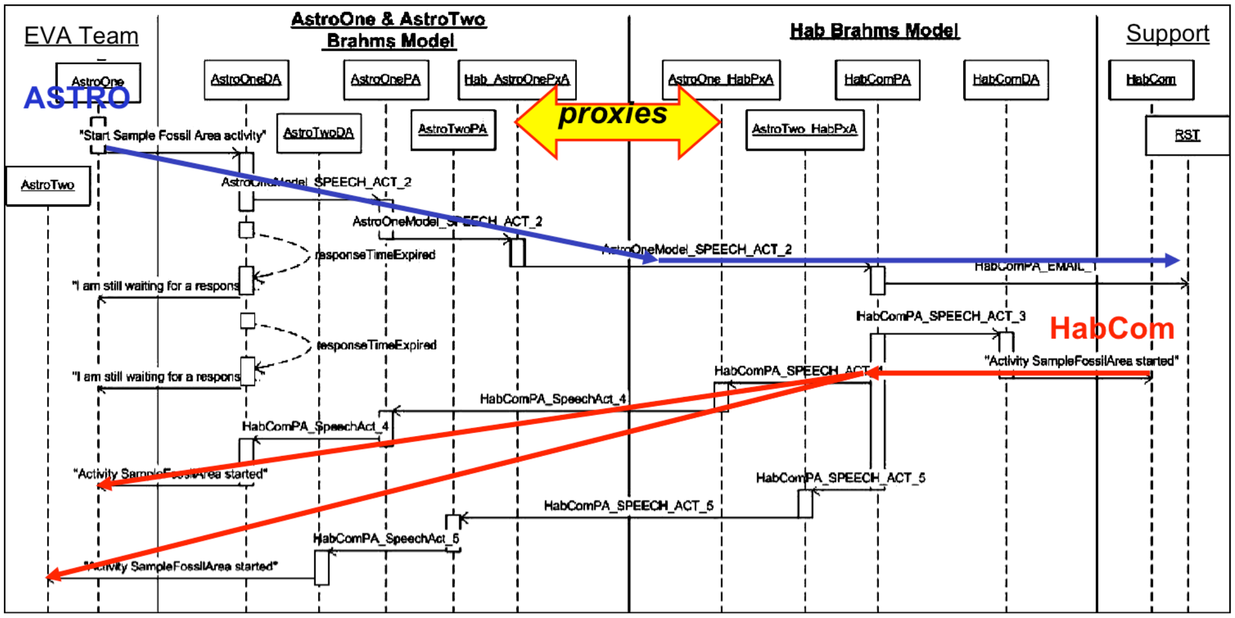

Figure 8.4. MDRS 2003 Example of Distributed Communications Using Proxy Agents. Initial design of MAA enabled agents to communicate with agents on another

platform by communicating with a local proxy for remote agents, e.g., Hab_AstroOnePxA located on the AstroOne platform was the proxy agent for the HabCom Agent located on the HabCom platform.

In this example shown in Figure 8.4, Astronaut One informs his personal agent (AstroOnePA) that he is ready to start sampling a particular area, referring to a named activity in the EVA plan. This statement is formalized as a SpeechAct[13] communicated from RIALIST (via the copy of the RIALIST CA running on Astronaut One's computer, called the AstroOneDA) to AstroOnePA. The AstroOnePA agent communicates this information to the HabComPA (via the proxy, Hab_AstroOnePxA), which sends an email to the remote science team to inform them that this activity has begun (keep in mind that the point of every MA field prototype was to demonstrate capabilities, not to advocate particular EVA or support protocols). HabCom then creates a new SpeechAct to inform the Astronaut that the information had been appropriately logged and conveyed. This SpeechAct follows a reverse path to the AstroOnePA via the AstroOne_HabPxA proxy, which upon receiving such a communication, the AstroOnePA is designed to transmit the SpeechAct to the RIALIST CA, causing a computer-generated voice to utter the statement through the astronaut's headphone. Simultaneously, the HabComPA has communicated the same utterance to Astronaut Two, so she would be informed as well that their common activity had begun (in this EVA the astronauts are by agreement working together; in other field tests the astronauts performed different tasks within a common EVA plan). Finally, notice that by design the AstroPA kept Astronaut One informed while it was waiting for a response for HabCom; again, the existence and frequency of these reminders are part of the workflow system's design, not the MAA itself, and were changed through experience, with the preference being to allow each astronaut to configure, enable, and disable each kind of alert.

In this example, on receiving the voice command from RIALIST, the AstroOneDA created the following SpeechAct:

```
SpeechAct(
        sender: AstroOne
        receiver: HabCom
        messageType: "request"
        messageAction: "StartActivity"
        messageSubject: "SampleFossilArea"
        replyTo: AstroOneDA)
```

HabCom transmitted the following SpeechAct in response to AstroOnePA:

[13] The format shown here was actually introduced in MDRS 2004. For Desert-RATS 2002 and MDRS 2003, speech acts were represented in the Brahms language as an attribute "speechact" that pointed to a Brahms object whose "beliefs" represented the content of the speech act (Sierhuis, 2001). Starting with MDRS 2004, SpeechActs were introduced into the Brahms language itself, as shown in this example. Consequently, the syntax of the MDRS 2003 implementation was more complicated than is shown here, but represented the same information.

```
SpeechAct(
                sender: HabCom
                receiver: AstroOnePA
                messageType: "inform"
                messageAction: "StartActivity"
                messageSubject: " SampleFossilArea "
                responseTo: <Request SpeechAct>)
```

Here the <Request SpeechAct> identifies this as a response to the initial request, and enables the AstroOnePA to detect that the request has been processed. Accordingly the AstroOnePA passes the speech act on to the RIALIST CA:

```
    SpeechAct(
                sender: HabCom
                receiver: AstroOneDA
                messageType: "inform"
                messageAction: "StartActivity"
                messageSubject: " SampleFossilArea "
                responseSpeech: "Activity SampleFossilArea started"
                responseTo: <Request SpeechAct>)
```

Starting in the MDRS 2005 configuration, tracking open requests was handled by the Plan Assistant group, which had an agent instance running on each platform (e.g., AstroOnePlanAssistant). On receiving a request (of the type that required tracking), a personal agent would convey the SpeechAct to the corresponding Plan Assistant agent, which would create a "task" object and schedule the task on the Task Queue. This enabled astronauts and components to asynchronously create multiple requests (that is, not waiting for a request to be processed before making another request). The Plan Assistant processed the queue using simple FIFO ordering, conveying the request to the appropriate agent specified in the SpeechAct. In particular, communications from AstroOnePA to HabCom would go through the Plan Assistant, which the HabCom would pass on to its own Plan Assistant. This design enabled the primary workflow agents (astronaut personal assistants and HabCom) to be very responsive in receiving and acknowledging requests, while processing them sequentially as time allowed. The design also allowed the overall workflow system to proceed robustly, enabling wireless communications to be degrade or be temporarily lost, and for agents to use processing time required to carry out a task.

As mentioned, the RIALIST speech system is implemented using the OAA framework and doesn't use CORBA. More typically, MA components such as the Scout rover used CORBA. For example, here is the speech act requesting Scout to follow an astronaut, created in response to "Scout, follow me" (or another named astronaut).

```
SpeechAct(
        sender: AstroOne
        receiver: ScoutPA
        messageType: "request"
        messageAction: "FollowMe"
        messageSubject: "AstroOne"
        replyTo: AstroOneDA)
```

The ScoutPA conveys this speech act to the ScoutCA, which calls a CORBA method provided by Scout's API. This method is essentially a CORBA object that calls a C++ method in Scout software to perform the intended action. The CORBA object includes a CallBack method for Scout's API to return the answer (status X = ok; X is turned on, etc). The ScoutCA then creates a reply SpeechAct and sends it back to the ScoutPA, which returns it to the AstroPA, which returns it to the RIALIST CA with the utterance, "Scout started following Astronaut One." The personal agents accordingly tell their Plan Assistant agents that the task is complete.

The examples given here extend directly to all of the functionalities demonstrated in the MA field experiments, including logging science data, naming locations, downloading photographs from a camera, taking panoramas, and modifying the EVA plan.

9 Bibliography

Alena, R. 2010. LSS Software Architecture Study: Real-time Avionics Middleware Architecture (Phase 2 Trade Study). NASA Ames Research Center.

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. *American Institute of Aeronautics and Astronautics 1st Space Exploration Conference,* 31 Jan-1 Feb, 2005, Orlando, FL. NASA Technical Publication 2007–214554.

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. Multiagent 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. 2010a. Collaborative Infrastructure Conceptual Overview. TM10-0001 NASA Ames Research Center, unpublished.

Clancey, W.J., Sierhuis, M., Nado, R., van Hoof, R., Lowry, M., Jones, G., & Dvorak, D. 2010b. *Lunar Surface Systems Software Open Architecture Study*. NASA Technical Publication 2007–216041.

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 (SMC-IT),* August 2011, Palo Alto, pp. 85-92.

CxP 70022-01, 2006. Constellation Program Command, Control, Communication, and Information (C3I) Interoperability Standards Book, Volume 1: Interoperability Specification. NASA Baseline Report, 18 December.

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.

Freund, T. & Niblett, P. 2008. Enterprise Service Bus (ESB) Interoperability Standards. IBM. Available: *http://www.ibm.com/developerworks/library/specification/ws-esb-interop/index.html*

HIMSS-Electronic Health Record Vendors Association (EHRVA). 2005. White paper on interoperability. Available: *http://www.himssehra.org/docs/EHRVAExpandedPositionStatementfinal042905.pdf.*

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.

Jennings, N. R., Sycara, K., and Wooldridge, M., "A Roadmap of Agent Research and Development," Autonomous Agents and Multiagent Systems, Vol. 1, 1998, pp. 7-38.

Johnson, A. W., Newman, D. J., Waldie, J. M., Hoffman, J. A., "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. 2010. *An Integrated EVA Mission Planner for Future Planetary Exploration*, M.S. thesis, Dept. of Aeronautics and Astronautics, Massachusetts Institute of Technology. Available: http://dspace.mit.edu/handle/1721.1/59560

Kukla, C. D., Clemens, E. A., Morse, R. S., and Cash, D. 1992. "Designing effective systems: A tool approach." In P.S. Adler and T.A. Winograd (eds.), *Usability: Turning Technologies into Tools,* Oxford University Press, New York, pp. 41-65.

Malin, J. 1999. Using Hybrid Modeling for Testing Intelligent Software for Lunar-Mars Closed Life Support. *Journal of Modeling and Simulation-e* 9, Available: *http://www.tms.org/pubs/journals/JOM/9909/Malin/Malin-9909.html*

Moses, J. 2010a. "Architecting engineering systems." In Ibo van de Poel and David E. Goldberg, (Eds.) *Philosophy and Engineering: An Emerging Agenda, Philosophy of Engineering and Technology Series,* Vol. 2, 1st Edition, pp. 275-284. New York: Springer.

Moses, J. 2010b. "Complexity, Flexibility and Layered Architectures," NECSI and MIT ESD Seminar Series, MIT, December 17.

Muscettola, N., Nayak, P. P., Pell, B., & Williams, B. C. 1998. Remote Agent: to boldly go where no AI system has gone before. *Artificial Intelligence* 103 (1-2) 5:47.

Network Centric Operations Industry Consortium (NCOIC). 2009. NIFTM Solution Description Reference Manual (NSD-RM), version 2.1, November. Available: *https://www.ncoic.org/technology/deliverables/nif/*

Nii, H.P. 1986a. Blackboard systems. *AI Magazine* 7(2), 38–53.

Nii, H.P. 1986b. Blackboard systems. *AI Magazine* 7(3), 82–106.

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*.

Peña, J., Hinchey, M. G., Ruiz-Cortés, A., and Trinidad, P. 2007. Building the Core Architecture of a NASA Multiagent System Product Line. In Lin Padgham and Franco Zambonelli (Eds.) *Agent-Oriented Software Engineering VII,* 7th International Workshop, AOSE 2006, Hakodate, Japan, May 2006. Berlin: Springer, Lecture Notes in Computer Science, Volume 4405, pp. 208-224.

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

Warren, S., Craft, R. L., Parks, R. C., Gallagher, L. K., Garcia, R. J., & Funkhouser, D. R. 1999. A proposed information architecture for telehealth system interoperability. *Proceedings of the First Joint BMES/EMBS Conference.* Sandia National Labs, Albuquerque, NM. Available:
http://www.osti.gov/bridge/product.biblio.jsp?osti_id=5692

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

Wooldridge, M., and Jennings, N. R. 1995. "Intelligent Agents: Theory and Practice." *Knowledge Engineering Review*, 10(2), 115-152.